Two-Grid Iterative Methods for Ill-Posed Problems

LYNGBY 2000
EKSAMENSPROJEKT
NR. 27/00

IMM

Trykt af IMM, DTU

Preface

This report was written at Department of Mathematical Modelling (IMM) at the at the Technical University of Denmark (DTU). It is the result of an exam project and amounts to 35 ects points out of 300 required as a part of obtaining the Masters Degree in Engineering at DTU.

The work has been done under supervision of professor Per Christian Hansen whom I'll thank for his support and many suggestions during the making of this thesis.

Michael Jacobsen, c958319
Kgs. Lyngby, September 4, 2000.

Abstract

Two-grid methods for regularized inverse problems, proposed by Hanke and Vogel [Numerische Matematik, 83:385–402, 1999], are investigated. The derivation of the methods are treated in detail and the algorithms are tested extensively. An important quantity is a subspace splitting and appropriate subspace splittings is discussed. The algorithms are applied to both small problems in order to investigate the impact on, e.g. the condition number and to large problems in order to see the results for some real world problems.

Keywords: conjugate gradients, preconditioning, ill-posed problem, ill-conditioned, regularization, two-grid.

Resumé

To-nivau metoder til regulariserede inverse problemer, som foreslået af Hanke og Vogel [Numerische Matematik, 83:385–402, 1999], er undersøgt. Udledningen af metoderne er gennemgået i detaljer, og algoritmerne er testet grundigt. I algoritmerne indgår en underrumsopdeling, og flere anvendelige opdelinger er undersøgt. Algoritmerne er anvendt i forbindelse med løsning af både små og store problemer for både at få overskuelige resultater i forbindelse med for eksempel konditionstallet og for at se resultater for realistiske problemer.

Notation

The notational rules used in this thesis are described below and a summary is found in Table 1 along with other reserved symbols.

Matrices are represented with bold capital letters, e.g., $\mathbf{A}$ and $\mathbf{C}$. A column of a matrix is written with a lower case bold letter with an index, e.g., the ith column of $\mathbf{A}$ is $\mathbf{a}_i$. The Matlab notation $\mathbf{A}_{1:k}$ is used to denote the matrix formed by the first k columns of $\mathbf{A}$. A specific element of a matrix is written with non-bold lower case character, e.g., a_{ij} denotes the element on the ith row jth column. If the matrix is diagonal only one index is used. As a guideline, an upper case italic non-bold characters denotes an operator in some continuous case.

A vector is represented with a bold lower case letter, e.g., $\mathbf{r}$. A specific element of a vector is written with a non bold letter with index, i.e., the ith element of $\mathbf{r}$ is r_i. A subvector corresponding to some block decomposition is written $\mathbf{r}_1$.

If a scalar, vector or matrix changes during some kind of iteration and the iteration number is of importance the iteration number appears within parentheses, e.g. $\mathbf{e}_{(i)}$.

A subspace is represented using capital "caligraphic" characters like $\mathcal{V}$ and $\mathcal{W}$. In the same manner the null space and the range of a matrix is denoted $\mathcal{N}(\mathbf{A})$ and $\mathcal{R}(\mathbf{A})$ respectively.

Symbol	Meaning	Ref. Page
$\mathbf{K}$	Discretized kernel matrix $(m \times n)$	
$\mathbf{L}$	Discretized regularization matrix $(p \times n)$	
$\mathbf{A}$	Normal equation coefficient matrix $\mathbf{A} = \mathbf{K}^T\mathbf{K} + \alpha\mathbf{L}^T\mathbf{L}$, $(n \times n)$	
$\mathbf{M}$	Normal equation coefficient matrix $\mathbf{M} = \mathbf{L}^T\mathbf{L}$ $(n \times n)$	
$\mathbf{N}$	Preconditioning matrix $(n \times n)$	
$\mathbf{I}_n$	Identity matrix $(n \times n)$	
$\mathbf{0}_{m,n}$	Matrix with zeros elements $(m \times n)$	
$\mathcal{K}(\mathbf{A}, \mathbf{q}, i)$	Krylov space	(3.1) 17
$\mathbf{K}(\mathbf{A}, \mathbf{q}, i)$	Krylov matrix	(3.2) 17
$\mathbf{U}$	Left singular matrix	(2.7) 10

Symbol	Meaning	Ref.	Page
$\mathbf{\Sigma}$	Diagonal matrix of singular values	(2.7)	10
σ_i	The ith singular value	(2.7)	10
$\mathbf{V}$	Right singular matrix	(2.7)	10
$\mathcal{N}(\cdot)$	The null-space of $\cdot$		
$\mathcal{R}(\cdot)$	The range of $\cdot$		
$\\|\mathbf{x}\\|_2$	The 2-norm ($\sqrt{\mathbf{x}^T\mathbf{x}}$)		
$\\|\mathbf{x}\\|_{\mathbf{A}}$	The $\mathbf{A}$-norm ($\sqrt{\mathbf{x}^T\mathbf{A}\mathbf{x}}$)	(3.12)	24
$\mathrm{cond}(\mathbf{A})$	The condition number ($\\|\mathbf{A}\\|\\|\mathbf{A}\\|$)		
$\otimes$	Kronecker product	(5.3)	75
δ_{ij}	Kronecker delta	(2.6)	10
α	Tikhonov regularization parameter		8
λ	An eigenvalue		
$(\cdot,\cdot)$	Inner product	(2.5)	10
$\mathbf{A}^\dagger$	The pseudo inverse	(2.8)	11
$\mathrm{vec}(\cdot)$	Columnwise stacking of columns		
$\mathrm{diag}(\cdot)$	Diagonal matrix constructed from $\cdot$		
$\mathcal{V}_k$	Coarse subspace (k-dimensional)		34
$\mathcal{W}_k$	The ($\mathbf{L}^T\mathbf{L}$-orthogonal) complementary subspace to $\mathcal{V}_k$ ($n-k$-dimensional)		34
$\mathbf{\Phi}$	Matrix with columns spanning $\mathcal{V}_k$ ($n \times k$)	4.2.1	34
$\mathbf{\Psi}$	Matrix with columns spanning $\mathcal{W}_k$ ($n \times n-k$)	4.2.1	34
$\gamma_{\mathbf{\Phi}}$	Operator norm in restricted subspace $\mathcal{V}_k$	(4.70)	57
$\gamma_{\mathbf{\Psi}}$	Operator norm in restricted subspace $\mathcal{W}_k$	(4.71)	57

Table 1: *Symbols and notation*

Contents

List of Algorithms

List of Tables

List of Figures

Chapter 1

Introduction

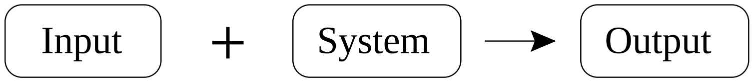
The main reason for this lack of discourse seems to be that the discussion of regularization techniques in the literature (...) is usually phrased in terms of functional analytic language, geared towards infinite-dimensional problems. (...). This tends to make the treatments unduly application specific and clutters the simplicity of the arguments with irrelevant details and distracting notation. Neumaier [22].

Some problems are harder than others. This report deals with very hard problems which in a sense are unsolvable. However, as we shall see, we are able to obtain a possible solution to the problems by using a mathematical trick called regularization. But first we will introduce the meaning of "ill-posed problems" and "preconditioning" which play an important role in this thesis. The following is a short mostly non-mathematical introduction to these two concepts.

1.1 Ill-Posed and Inverse Problems

An ill-posed problem is in the linear algebra setting defined by a system of equations that often originates from an ill-posed inverse problem with a huge condition number. A huge condition number indicates that rounding errors or other errors will seriously influence the result. The concept of a "problem" in this context is illustrated in Figure 1.1. In the case of a forward problem the input and system are given and the problem is to find the output.

The inverse problem has two variants. The most common is that the system and output are known and one wishes to find the input. The case where the unknown is the system is also denoted an inverse problem.



$$Kx = y$$

$$\mathbf{K}\mathbf{x} = \mathbf{y}$$

Figure 1.1: *The structure of a problem in a graphical form, in operator form and in the linear algebra form.*

An inverse problem can also be explained using an example from the popular spare time occupation TV watching. On TV most quiz shows ask a question and the participant must answer — a forward problem. In Jeopardy the answer is given and the participants must "answer" with the question — an inverse problem.

Returning to a more scientific setting we have the standard example atmospheric blur. In this example the system is the atmosphere which blurs the light from the stars (the input). The output is measured by a camera and the inverse problem is to determine what the stars really look like. A mathematical approach of finding the solution is treated in this thesis, while the well known Hubble Telescope takes another approach and simply removes the problem by moving out of the atmosphere[1]. However a space telescope is a rather expensive device and a more earth-bound approach could in many cases be cheaper. Furthermore for many problems it is not possible to remove the problem. Helios seismology is a good example because it is impossible (with current technology — never say never) to travel to the sun and drill holes in it in order to study its internal structure. So the study of inverse problems is very relevant in some areas.

Some inverse problems are hard while others are easy to solve. It is obvious that the easy solvable problems are less interesting (we just solve them) than the (very) hard ones, which we call *ill-posed*. An ill-posed

[1] A problem with the optic parts of Hubble did unfortunately create an unwanted inverse problem, which at first was dealt with using mathematics but the telescope was later equipped with "glasses".

problem is ill-conditioned and neither a simple nor a complicated reformulation of the problem will improve the ill-conditioned problem significantly. An ill-conditioned problem is on the other hand not necessarily an ill-posed problem because a reformulation sometimes does help. In a strict mathematical setting we are not able to solve an ill-posed problem and get the correct solution. However using a priori knowledge we are able to get an answer that, hopefully, is close to the correct solution. One of the tools in this category is regularization which is used to tame the solution and keep it within reasonable limits.

We now return to the case of atmospheric blur to explain the ill-posed nature of this particular problem. We have outside the atmosphere a picture of some stars. This picture portrays point objects because of the enormous distances. These points get blurred on the way through the atmosphere, i.e., the edges are smoothened. This picture arrives to a telescope which takes a picture with some unavoidable measurement errors (noise). The forward model smoothened the edges so the inverse must naturally sharpen the image. However, there is no way to separate measurement errors from the data and the noise is therefore also sharpened. Noise gives sharp edges in the blurred image and the inverse operation furthermore sharpens the noise. In many cases the results is that the sharpened noise dominates the real image.

Much work on ill-posed problems has been done since the early works by Phillips [27] and Tikhonov [33]. A more recent work is by Hansen [16] that treats ill-posed problems in a linear algebra setting very similar to the style in this thesis.

1.2 Iterative Methods

The word preconditioning hints that it is something you do before and it is something you do to help the condition of the problem. Preconditioning is mostly used to help iterative methods such as conjugate gradients.

The counterpart to iterative methods are direct methods. Direct methods are characterized by some kind of factorization. The well known Gaussian is nothing more than an LU factorization with forward and backward substitution. A direct method finds a solution — right or wrong — in "one" step and normally we do not get any approximate solutions during this single step.

The downside to direct methods becomes apparent when the problems get large. A factorization of a large matrix is generally very expensive in terms of computing time and memory consumption.

Iterative methods are characterized by using repeated steps in order to find the solution. At each step a new solution is generated which hopefully converges to the correct solution. Furthermore one iteration is normally cheap in terms of computing time compared to a direct method. If the iterations converge fast it possible to stop after just a few iterations and still have a good approximation to the solution.

Many iterative methods also have the advantage that the system matrix $\mathbf{A}$ itself is not important but rather its effect on a vector. Hence it is only necessary have some function that computes the matrix-vector product (a black box).

Iterative methods make it possible to work with larger problems than the direct methods. This fact combined with scientists' ability to enlarge and extend any given problem to any given size we see a large interest for iterative methods. Hence iterative methods are both treated in specialized books by, e.g. Axelsson [1]; Smith, Bjørstad and Gropp [32]; Barret et al. [2]; and Hageman and Young [10] but also as important parts of more general books on linear algebra by, e.g., Golub and Van Loan [7]; Demmel [5]; and Trefethen and Bau [34]. An example of the special combination of ill-posed problems and iterative methods is by Hanke [11].

1.3 Thesis Overview

This chapter has given the reader a brief overview of what an ill-posed problem is. The concepts and theory of ill-posed problems are further discussed in Chapter 2 along with the mathematical tool regularization that is used to solve these the problems.

Chapter 3 describes the iterative method named conjugate gradients. Convergence is treated, which naturally leads to the concept of preconditioning. Preconditioning is a technique to modify a problem in order to accelerate convergence.

Ill-posed problems have special properties which the two-grid preconditioners proposed by Hanke and Vogel [13], [12] try to exploit. Chapter

4 takes the reader through the construction of the algorithms as well as indications on the proper use of the methods.

An important part of the two-grid preconditioners is a division of the solution space into two subspaces. Chapter 5 gives an overview of some subspace types suitable for use in our methods.

In order to investigate, illustrate and support the mathematical theory, Chapter 6 contains tests and experiments with the algorithms. Numerical experiments are performed on a small suite of test-problems of varying difficulty and size. This chapter along with Chapters 4 and 5 form the essence of two-grid methods and therefore also this thesis.

Finally, in Chapter 7, the results and observations are summed up and fields for further investigations are listed.

Implementations of the algorithms are listed in the appendix along with descriptions of input and output parameters. Documentation on the test procedures performed to verify the correctness of the algorithms with respect to the mathematical deductions in Chapter 4 is also located in the appendix.

Chapter 2

Ill-Posed Problems and Regularization

For a long time mathematicians felt that ill-posed problems cannot describe real phenomena and objects. However, we shall show in the present book that the class of ill-posed problems includes many classical mathematical problems and, most significantly, that such problems have important applications. Tikhonov [33].

This thesis treats the problem of solving linear systems with large condition numbers. In all introductory courses on numerical linear algebra one learns that a large condition number means that the result should not be trusted. A common advice is to reformulate the problem to get a better conditioned system of equations, i.e., with a smaller condition number. However for some problems it is not possible to do so because the underlying problem is ill-posed.

The classical example of an inverse ill-posed problem is the Fredholm integral equation of the first kind with a square integrable kernel[1]

$$\int_I K(s,t)x(t)dt = y(s). \tag{2.1}$$

Square integrable means that the kernel $K(s,t)$ fulfill

$$||K||^2 = \int_I \int_I |K(s,t)|^2 dtds < \infty.$$

[1]An example of this type is the atmospheric blur problem.

In a functional analysis setting equation (2.1) is written in operator form

$$Kx = y, \tag{2.2}$$

where K is a compact Hilbert-Schmidt operator. The inverse problem — finding x from K and y — is problematic due to K being compact. A compact K implies that the inverse K^{-1} is unbounded (if it exists!). The result is that the solution x might not have a continuous dependence on y [26, Proposition 5.9].

This fact is mirrored in a linear system of equations $\mathbf{Kx} = \mathbf{y}$ in the sense that the condition number is large. If the mapping K^{-1} had been continuous then a little modification in y would result in a bounded and controllable change in x. When we discretize the integral equation we are bound to make errors and during the solution of the discretized system we are going to add even more errors. Furthermore $\mathbf{y}$ often originates from some kind of measurements which seldom or more likely never are measured exact.

2.1 Regularization

Regularization is a common name for methods to bound the unbounded K^{-1}. One of the best known regularization methods is *Tikhonov regularization* which minimizes a linear combination of $||Kx - y||_2^2$ and $||Lx||_2^2$, that is

$$x = \min_x \left(||Kx - y||_2^2 + \alpha ||Lx||_2^2 \right), \tag{2.3}$$

where α is a parameter which controls the degree of regularization. The operator L introduces a "penalty" if x behaves undesirable. The simple choice is to let L be the identity and hereby the regularization restricts the norm of x. The choice $\alpha = 0$ gives us the possibly unbounded result of $K^{-1}y$ while choosing "$\alpha = \infty$" yields a result result $x \in \mathcal{N}(L)$, i.e., in the null space of L. In the special case $L = I$ we get $y = 0$.

Tikhonov regularization leads to three problems which all must be addressed to find the best result:

- How to solve the minimization problem (2.3).
- How to choose α. If α is chosen to large we restrict the solution to much and vice versa.

- How to choose L.

The main subject of this thesis is the first problem — how to solve the system, while the second and third only occupy a few pages. On the subject of the choice of α consult [16] and references therein. The choice of L depends on a priori information on the physics of the system and solution. Therefore is the choice is often obvious.

2.1.1 Regularization of Discrete Problems

Most regularization methods are most easily described in the case where the problem has been discretized into a system of linear equations.

Tikhonov regularization can be moved from the continuous case to the discrete case without problems. Using some kind of discretization the operators K and L become matrices and x and y turn into vectors.

$$\min_x \left(\|\mathbf{Kx}-\mathbf{y}\|_2^2 + \alpha\|\mathbf{Ly}\|_2^2\right) = \min_x \left\| \begin{bmatrix} \mathbf{K} \\ \sqrt{\alpha}\mathbf{L} \end{bmatrix} \mathbf{x} - \begin{bmatrix} \mathbf{y} \\ \mathbf{0} \end{bmatrix} \right\|_2^2,$$

on which the normal equations can be applied

$$(\mathbf{K}^T\mathbf{K} + \alpha\mathbf{L}^T\mathbf{L})\mathbf{x} = \mathbf{K}^T\mathbf{y}$$

Before we proceed we need to introduce an indispensable tool for the solution and diagnosis of ill-posed problems; the singular value decomposition.

2.2 The Singular Values and Vectors

The singular value decomposition (SVD) is a valuable tool in the quest of solving ill-posed problems. We shall see how the SVD gives an insight into the fundamental problems stemming from an ill-posed problem. The following discussion is based on Chapters 1 and 2 of [16] and Chapter 1 of [19]. Once again we start in the continuous case before we move into the discrete world. In order to expose the concepts and ideas rather than the details we have assumed $L = I$. The general case is more complicated but reveals nothing important in this context. In fact it is possible to transform the general problem with $L \neq I$ into the "standard form" with $L = I$, see [16, Section 2.3.1].

The Continuous Case

In functional analysis we have a tool named the singular value expansion (SVE) which is a Fourier-like expansion of the kernel

$$K(s,t) = \sum_{i=1}^{\infty} \mu_i u_i(s) v_i(t),$$

where the functions u_i and v_i are called the singular functions. They must be mutually orthogonal in the sense that

$$(u_i, u_j) = (v_i, v_j) = \delta_{ij},$$

where the inner product is the usual (assuming all functions to be real)

$$(\phi, \varphi) = \int_I \phi(t)\varphi(t)dt, \tag{2.5}$$

and δ_{ij} denotes the usual Kronecker delta

$$\delta_{ij} = \begin{cases} 0 & i \neq j \\ 1 & i = j \end{cases}. \tag{2.6}$$

The Discrete Case

The SVD is to matrices what the SVE is to kernels of Fredholm integral equations. The thin SVD of a matrix $\mathbf{K} \in \mathbb{R}^{m\times n}$, $m \geq n$ is a decomposition of the form

$$\mathbf{K} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = \sum_{i=1}^{n} \mathbf{u}_i \sigma_i \mathbf{v}_i^T, \tag{2.7}$$

where $\mathbf{U} = [\mathbf{u}_1, \ldots, \mathbf{u}_n] \in \mathbb{R}^{m\times n}$ and $\mathbf{V} = [\mathbf{v}_1, \ldots, \mathbf{v}_m] \in \mathbb{R}^{n\times n}$ are matrices with orthonormal columns. The matrix $\mathbf{\Sigma}$ is a diagonal matrix with elements σ_i ordered non-increasing. If $m < n$ the SVD is found by decomposing $\mathbf{K}^T$ followed by an interchange of $\mathbf{U}$ and $\mathbf{V}$.

Worth noticing is that the SVD is a close cousin to the eigenvalue decomposition because $(\mathbf{K}^T\mathbf{K}) = \mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^T$ and $(\mathbf{K}\mathbf{K}^T) = \mathbf{U}\mathbf{\Sigma}^2\mathbf{U}^T$. If a matrix $\mathbf{A}$ is symmetric positive definite (SPD), i.e., $\mathbf{x}^T\mathbf{A}\mathbf{x} > 0$ for $x \neq \mathbf{0}$, we can calculate a Cholesky decomposition $\mathbf{A} = \mathbf{C}\mathbf{C}^T$ and from the SVD of the Cholesky factor $\mathbf{C} = \mathbf{U}\hat{\mathbf{\Sigma}}\mathbf{V}^T$ we get

$$\mathbf{A} = \mathbf{C}\mathbf{C}^T = \mathbf{U}\hat{\mathbf{\Sigma}}\mathbf{V}^T\mathbf{V}\hat{\mathbf{\Sigma}}\mathbf{U}^T = \mathbf{U}\hat{\mathbf{\Sigma}}^2\mathbf{U}^T,$$

and we see that the SVD of a SPD matrix $\mathbf{A}$ is equal to an eigenvalue decomposition with $\lambda_i = \sigma_i^2$ and possibly some sign changes in the vector pairs $(\mathbf{u}_i, \mathbf{v}_i)$.

The Pseudo Inverse

The SVD can also be used to define the *pseudo inverse* that extends the notion of inverse matrices to singular and even rectangular matrices. Let $\mathbf{K} \in \mathbb{R}^{m\times n}$ have the SVD $\mathbf{K} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = \sum_{i=1}^n \mathbf{u}_i\sigma_i\mathbf{v}_i^T$. The highest index i, for which $\sigma_i \neq 0$, defines the rank of $\mathbf{K}$ denoted rank($\mathbf{K}$). Let $r =$ rank($\mathbf{K}$) and the pseudo inverse is then defined as

$$\mathbf{K}^\dagger = \sum_{i=1}^{\mathrm{rank}(\mathbf{K})} \mathbf{v}_i\sigma_i^{-1}\mathbf{u}_i^T = \mathbf{V}\begin{pmatrix} \mathbf{\Sigma}_{1:r,1:r}^{-1} & \mathbf{0}_{r,n-r} \\ \mathbf{0}_{n-r,r} & \mathbf{0}_{n-r,n-r} \end{pmatrix}\mathbf{U}^T. \tag{2.8}$$

Notice that the inverse and pseudo inverse are equal if $\mathbf{K}$ is square and nonsingular. Furthermore the pseudo inverse applied to a vector results in the least square solution:

$$\begin{aligned}
\mathbf{x}_{\mathrm{LS}} &= (\mathbf{K}^T\mathbf{K})^{-1}\mathbf{K}^T\mathbf{b} \\
&= (\mathbf{V}\mathbf{\Sigma}\mathbf{U}^T\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T)^{-1}\mathbf{V}\mathbf{\Sigma}\mathbf{U}^T\mathbf{b} \\
&= (\mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^T)^{-1}\mathbf{V}\mathbf{\Sigma}\mathbf{U}^T\mathbf{b} \\
&= \mathbf{V}(\mathbf{\Sigma}^\dagger)^2\mathbf{V}^T\mathbf{V}\mathbf{\Sigma}\mathbf{U}^T\mathbf{b} \\
&= \mathbf{V}\mathbf{\Sigma}^\dagger\mathbf{U}^T\mathbf{b} \\
&= \mathbf{K}^\dagger\mathbf{b}.
\end{aligned}$$

The Singular Vectors

The right singular vectors $\mathbf{v}_i$ will play an important role in this thesis. It will be shown in a later chapter that a good knowledge of the right singular vectors will prove valuable in constructing a good two-grid method.

The singular vectors $\mathbf{v}_i$ tend to have more zero-crossings (more oscillations) as index increases, or in other words as σ_i decreases smaller. Unfortunately this is based on experience and it is perhaps impossible to prove [16]. However, for all problems discussed in this thesis the above mentioned property holds to the extend that is has been possible to calculate the SVD. That is, the property has not been verified for the larger problems.

2.2.1 Filter Factors

A solution can be characterized with respect to the SVD

$$\mathbf{x}_{\mathrm{reg}} = \sum_{i=1}^n f_i \frac{\mathbf{u}_i^T\mathbf{b}}{\sigma_i}\mathbf{v}_i, \tag{2.9}$$

where the coefficients f_i are called the *filter factors*. Comparing (2.9) with (2.8) we see the least square solution have $f_i = 1$ for all i. If we have an arbitrary solution the filter factors are found via

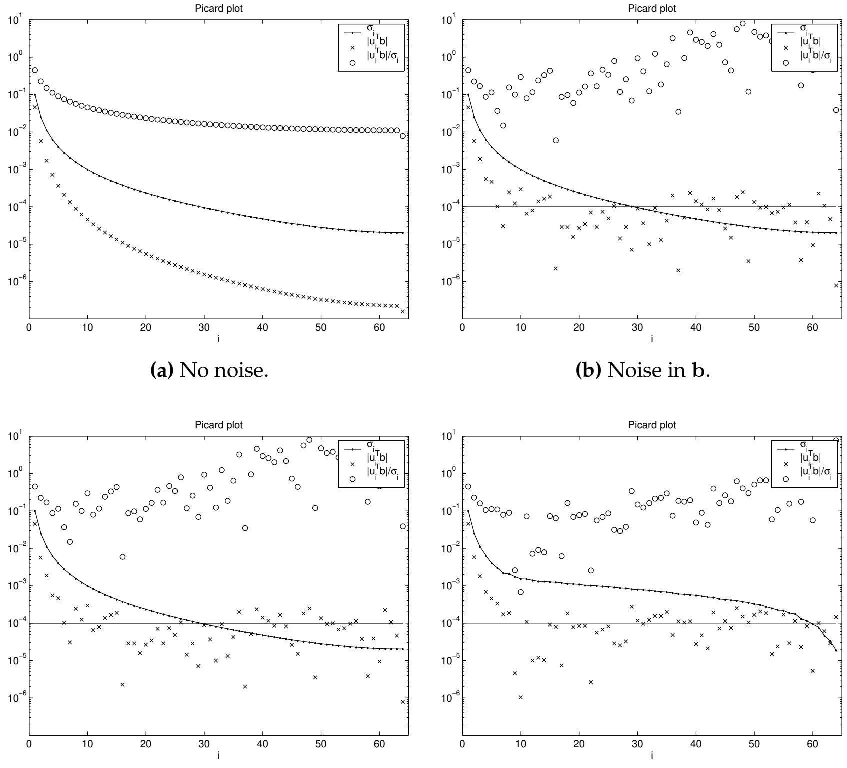
$$f_i = \frac{\mathbf{v}_i^T\mathbf{x}_{\mathrm{reg}}}{\mathbf{v}_i^T\mathbf{x}_{\mathrm{exact}}}, \quad \mathbf{v}_i^T\mathbf{x}_{\mathrm{exact}} \neq 0.$$

We now have the tools to describe the problems of ill-posed problems as well as the solution technique regularization.

2.2.2 Perturbations and the SVD

As already mentioned the main problem when solving an ill-posed problem is rounding errors and inaccuracies in the vector $\mathbf{b}$ and the matrix $\mathbf{A}$. The influence can be illustrated by Picard plots [15]. A Picard plot shows how the singular values, σ_i, the Fourier coefficients, $|\mathbf{u}_i^T\mathbf{b}|$, and the "solution coefficients", $|\mathbf{u}_i^T\mathbf{b}/\sigma_i|$, vary with the index.

Figure 2.1 shows a representative example with the cases of no noise, noise in $\mathbf{b}$, noise in $\mathbf{A}$ and noise in both $\mathbf{A}$ and $\mathbf{b}$. We observe that the coefficients with index $i > 10$ are disturbed the most while the components belonging to the larger singular values and their vectors are relatively unharmed.



(a) No noise. **(b)** Noise in $\mathbf{b}$. **(c)** Noise in $\mathbf{A}$. **(d)** Noise in both $\mathbf{A}$ and $\mathbf{b}$.

Figure 2.1: *Picard plots for test problem* deriv2. *The added noise is normally distributed with deviation 10^{-4} (indicated with a horizontal line).*

It is clear that the solution coefficients for $i > 10$ are "of target" and should somehow be avoided by the regularization method, while the remaining coefficients should be preserved.

A conceptually very simple regularization method is the truncated SVD (TSVD) which simply removes all information from the singular values and vectors from a given index and up. Hence the filter factors of the TSVD method are very simple

$$f_i = \begin{cases} 1 & i \le k \\ 0 & i > k \end{cases},$$

where k is a regularization parameter (the truncation).

The TSVD solution might seem a bit rough. The Tikhonov regularization filter factors

$$f_i = \frac{\sigma_i^2}{\sigma_i^2 + \alpha},$$

are more smooth, see Figure 2.2, and are in that respect maybe more attractive [16]. The equation shows that components belonging to singular values smaller than the square root of the regularization parameter, while those components with larger singular values get through the filter unharmed. We note that $\alpha = 0$ yields $f_i = 1$ (the least squares solution) and $\alpha = \infty$ yields $f_i = 0$ implying $\mathbf{x} = \mathbf{0}$.

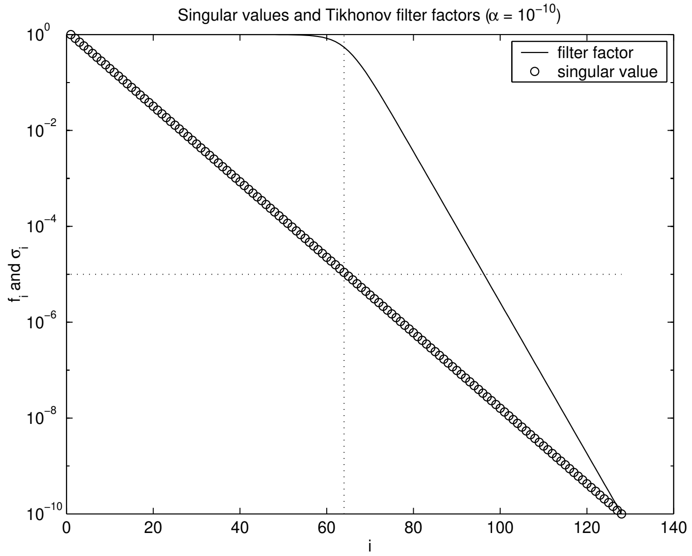


Figure 2.2: *Simulated Tikhonov filter factors. The horizontal dotted line shows the value of α and the vertical dotted line shows the index where $\alpha > \sigma_i^2$. The filter factors with index $i < 63$ are nearly one such that the corresponding solution components are unharmed. On the other hand the filter factors with index $i > 63$ decay rapidly and the corresponding solution components are damped.*

Chapter 3

Conjugate Gradients

The results indicate that the method is very suitable for high speed machines. Hestenes and Stiefel in the original article on Conjugate Gradients (1952)[18]

This chapter deals with an important iterative method named Conjugate Gradients (CG) and we leave ill-posed problems until Chapter 4. CG is a method which in its pure form solves $\mathbf{Ax} = \mathbf{b}$ when $\mathbf{A}$ is SPD. The CG method originates from the childhood of computers and was first presented in a paper by Hestenes and Stiefel [18].

CG is classified as a Krylov subspace method because the iterations takes place in the so called Krylov subspaces. A Krylov subspace is defined by a vector $\mathbf{q}$, a square matrix[1] $\mathbf{A}$, and an integer k by

$$\mathcal{K}(\mathbf{A}, \mathbf{q}, k) = \text{span}\left\{\mathbf{q}, \mathbf{Aq}, \mathbf{AAq}, \ldots, \mathbf{A}^{k-1}\mathbf{q}\right\}, \tag{3.1}$$

and the corresponding Krylov matrix is defined by

$$\mathbf{K}(\mathbf{A}, \mathbf{q}, k) = \left[\mathbf{q}, \mathbf{Aq}, \mathbf{AAq}, \ldots, \mathbf{A}^{k-1}\mathbf{q}\right]. \tag{3.2}$$

Many well known methods such as GMRES and LSQR are members of the Krylov subspace family but these are only a selection of the many others that have been developed since 1952 with CG as source of inspiration [8].

[1]In this chapter $\mathbf{A}$ is not necessarily $\mathbf{K}^T\mathbf{K} + \alpha\mathbf{M}$.

3.1 The Basics of Conjugate Gradients

CG is often derived as a minimization algorithm of the function $f(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}$:

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{Ax} - \mathbf{b}^T\mathbf{x}, \tag{3.3}$$

where $\mathbf{A}$ is SPD. The function has one minimum in which the gradient $\nabla f(\mathbf{x}) = \mathbf{Ax} - \mathbf{b}$ equals the zero vector. Thus a minimization of (3.3) also solves the linear system $\mathbf{Ax} = \mathbf{b}$. Notice how the negated gradient $-\nabla f(\mathbf{x}) = \mathbf{b} - \mathbf{Ax}$ equals the residual $\mathbf{r}(\mathbf{x})$. Before we proceed with the CG algorithm we introduce the steepest descent method which is a natural step stone on the way to an understanding of CG.

Steepest Descent

Steepest descent is a very simple iterative algorithm for minimization of (3.3). It iteratively performs line searches along a line $\mathbf{x}_{(i)} + \beta\mathbf{d}$ such that $f(\mathbf{x}_{(i)} + \beta\mathbf{d})$ is minimized. A minimization is achieved when the directional derivative with respect to β is zero for the new iterant $\mathbf{x}_{(i+1)} = \mathbf{x}_{(i)} + \beta\mathbf{d}$ (this follows from the fact that $\mathbf{A}$ is SPD)

$$\frac{d}{d\beta}f(\mathbf{x}_{(i+1)}) = \mathbf{f}'(\mathbf{x}_{(i+1)})^T\frac{d}{d\beta}\mathbf{x}_{(i+1)} = \mathbf{f}'(\mathbf{x}_{(i+1)})^T\mathbf{d} = 0. \tag{3.4}$$

Equation (3.4) tells us that β should be chosen so that $\mathbf{f}'(\mathbf{x}_{(i)})$ and $\mathbf{d}$ are orthogonal. Remember that $\mathbf{f}'(\mathbf{x}) = \mathbf{Ax} - \mathbf{b}$ and we have:

$$\begin{aligned}
(\mathbf{Ax}_{(i+1)} - \mathbf{b})^T\mathbf{d} &= 0 \\
(\mathbf{b} - \mathbf{A}(\mathbf{x}_{(i)} + \beta\mathbf{d}))^T\mathbf{d} &= 0 \\
(\mathbf{b} - \mathbf{Ax}_{(0)})^T\mathbf{d} - \beta(\mathbf{Ad})^T\mathbf{d} &= 0 \\
&\Updownarrow \\
\beta &= \frac{(\mathbf{b} - \mathbf{Ax}_{(0)})^T\mathbf{d}}{\mathbf{d}^T\mathbf{Ad}}.
\end{aligned}$$

Because $\mathbf{A}$ is SPD and the search direction is non-zero we are able to compute β.

Steepest descent method uses this formula, where the direction $\mathbf{d}$ in each iteration is taken to be the negated gradient (i.e., the steepest descent direction) $-\nabla f(\mathbf{x}) = \mathbf{b} - \mathbf{A}\mathbf{x} = \mathbf{r}(\mathbf{x})$. Hence steepest descent can be formulated as stated in Algorithm 1. The update of the residual $\mathbf{r}_{(i+1)}$ is achieved by taking $\mathbf{x}_{(i+1)} = \mathbf{x}_{(i)} + \beta^{(i)}\mathbf{r}_{(i)}$ pre-multiplying with $-\mathbf{A}$ and adding $\mathbf{b}$.

Algorithm 1 Steepest Descent. Solve $\mathbf{A}\mathbf{x} = \mathbf{b}$, $\mathbf{A} \in \mathbb{R}^{n\times n}$ and SPD.

$\mathbf{r}_{(0)} = \mathbf{b} - \mathbf{A}\mathbf{x}_{(0)}$
$i \leftarrow 0$
repeat
  $\beta^{(i)} = \frac{\mathbf{r}_{(i)}{}^T\mathbf{r}_{(i)}}{\mathbf{r}_{(i)}{}^T\mathbf{A}\mathbf{r}_{(i)}}$
  $\mathbf{x}_{(i+1)} = \mathbf{x}_{(i)} + \beta_{(i)}\mathbf{r}_{(i)}$
  $\mathbf{r}_{(i+1)} = \mathbf{r}_{(i)} - \beta\mathbf{A}\mathbf{r}_{(i)}$
  $i \leftarrow i + 1$
until some stop criterion

Steepest descent is an easily understood algorithm but unfortunately it has very bad convergence properties. Often steepest descent chooses to minimize in the same few directions over and over again [31].

3.1.1 Conjugate Gradients

The CG method remedies the shortcomings of steepest descent by forcing the search directions $\mathbf{d}_{(i)}$ to be $\mathbf{A}$-conjugate, that is $\mathbf{d}_{(i)}{}^T\mathbf{A}\mathbf{d}_{(j)} = 0$ if $i \neq j$ (also denoted $\mathbf{A}$-orthogonal). Furthermore the residuals are forced to be orthogonal which leads to the main point of the algorithm. We are working in an n-dimensional space and therefore the algorithm is guaranteed to end because the residual $\mathbf{r}_{(n+1)}$ must be the zero vector in order to be "orthogonal" to the n previous residuals. The algorithm is listed in Algorithm 2 and the above mentioned properties and a proofs of these are formalized in the following.

Theorem 1 *Assuming infinite precision Algorithm 2 satisfies: The residuals $\mathbf{r}_{(i)}$ are mutually orthogonal and the search directions $\mathbf{d}_{(i)}$ are mutually $\mathbf{A}$-orthogonal, that is*

$$\begin{aligned} \mathbf{r}_{(i)}{}^T\mathbf{r}_{(j)} &= 0 \quad \text{if } i \neq j, \qquad (3.5)\\ \mathbf{d}_{(i)}{}^T\mathbf{A}\mathbf{d}_{(j)} &= 0 \quad \text{if } i \neq j. \qquad (3.6) \end{aligned}$$

Algorithm 2 Basic Conjugate Gradients (CG). Solve $\mathbf{A}\mathbf{x} = \mathbf{b}$, $\mathbf{A} \in \mathbb{R}^{n\times n}$ and SPD.

1: $\mathbf{d}_{(0)} = \mathbf{r}_{(0)} = \mathbf{b} - \mathbf{A}\mathbf{x}_{(0)}$
2: $i \leftarrow 0$
3: **while** $\mathbf{r}_{(i)} \neq 0$ **do**
4:   $\beta_{(i)} = \frac{\mathbf{r}_{(i)}{}^T\mathbf{r}_{(i)}}{\mathbf{d}_{(i)}{}^T\mathbf{A}\mathbf{d}_{(i)}}$
5:   $\mathbf{x}_{(i+1)} = \mathbf{x}_{(i)} + \beta_{(i)}\mathbf{d}_{(i)}$
6:   $\mathbf{r}_{(i+1)} = \mathbf{r}_{(i)} - \beta_{(i)}\mathbf{A}\mathbf{d}_{(i)}$
7:   $\gamma_{(i+1)} = \frac{\mathbf{r}_{(i+1)}{}^T\mathbf{r}_{(i+1)}}{\mathbf{r}_{(i)}{}^T\mathbf{r}_{(i)}}$
8:   $\mathbf{d}_{(i+1)} = \mathbf{r}_{(i+1)} + \gamma_{(i+1)}\mathbf{d}_{(i)}$
9:   $i \leftarrow i + 1$
10: **end while**

Proof First we note that $\beta_{(i)} \neq 0$ and $\gamma_{(i)} \neq 0$ unless we have found the solution. The proof is done by induction. In the initial case we have $\mathbf{r}_{(0)}$, $\mathbf{d}_{(0)} = \mathbf{r}_{(0)}$, and $\mathbf{r}_{(1)}$. Equation (3.5) is satisfied because

$$\begin{aligned} \mathbf{r}_{(0)}{}^T\mathbf{r}_{(1)} &= \mathbf{r}_{(0)}{}^T(\mathbf{r}_{(0)} - \beta_i\mathbf{A}\mathbf{d}_{(0)}) \\ &= \|\mathbf{r}_{(0)}\|_2^2 - \frac{\mathbf{r}_{(0)}{}^T\mathbf{r}_{(0)}}{\mathbf{d}_{(0)}{}^T\mathbf{A}\mathbf{d}_{(0)}}\mathbf{r}_{(0)}{}^T\mathbf{A}\mathbf{d}_{(0)} \\ &= 0. \qquad (3.7) \end{aligned}$$

The first part of the induction step assumes, for some k, that the theorem holds for the set of residuals $\mathbf{r}_{(0)}, \ldots, \mathbf{r}_{(k)}$ and the set of search directions $\mathbf{d}_{(0)}, \ldots, \mathbf{d}_{(k-1)}$. We wish to add the search direction $\mathbf{d}_{(k)}$ defined by Algorithm 2 step 8 to this set. The first step is to find an expression of $\mathbf{d}_{(k)}$

using the known residuals

$$\begin{aligned}
\mathbf{d}_{(k)} &= \mathbf{r}_{(k)} + \gamma_{(k-1)}\mathbf{d}_{(k-1)} \\
&= \frac{\|\mathbf{r}_{(k)}\|_2^2}{\|\mathbf{r}_{(k)}\|_2^2}\mathbf{r}_{(k)} + \frac{\|\mathbf{r}_{(k)}\|_2^2}{\|\mathbf{r}_{(k-1)}\|_2^2}\mathbf{d}_{(k-1)} \\
&= \frac{\|\mathbf{r}_{(k)}\|_2^2}{\|\mathbf{r}_{(k)}\|_2^2}\mathbf{r}_{(k)} + \frac{\|\mathbf{r}_{(k)}\|_2^2}{\|\mathbf{r}_{(k-1)}\|_2^2}\left(\mathbf{r}_{(k-1)} + \frac{\|\mathbf{r}_{(k-1)}\|_2^2}{\|\mathbf{r}_{(k-2)}\|_2^2}\mathbf{d}_{(k-2)}\right) \\
&\vdots \\
&= \|\mathbf{r}_{(k)}\|_2^2 \sum_{j=0}^{k} \frac{\mathbf{r}_{(j)}}{\|\mathbf{r}_{(j)}\|_2^2}. \qquad (3.8)
\end{aligned}$$

Assuming $i \leq k$ we find from (3.8) and the orthogonality of the residuals $\mathbf{r}_{(0)}, \ldots, \mathbf{r}_{(k)}$ that

$$\begin{aligned}
\mathbf{r}_{(i)}{}^T\mathbf{d}_{(k)} &= \|\mathbf{r}_{(k)}\|_2^2 \sum_{j=0}^{k} \frac{\mathbf{r}_{(i)}{}^T\mathbf{r}_{(j)}}{\|\mathbf{r}_{(j)}\|} \\
&= \|\mathbf{r}_{(k)}\|^2. \qquad (3.9)
\end{aligned}$$

Now let $i < k$ and we use step 6 of the algorithm and the previous expression to show

$$\begin{aligned}
\mathbf{r}_{(i+1)}{}^T\mathbf{d}_{(k)} &= \mathbf{r}_{(i)}{}^T\mathbf{d}_{(k)} - \beta_{(k)}\mathbf{d}_{(i)}{}^T\mathbf{A}\mathbf{d}_{(k)} \\
&\Updownarrow \\
\|\mathbf{r}_{(k)}\|^2 &= \|\mathbf{r}_{(k)}\|^2 - \beta_{(k)}\mathbf{d}_{(i)}{}^T\mathbf{A}\mathbf{d}_{(k)} \\
&\Updownarrow \\
\mathbf{d}_{(i)}^T\mathbf{A}\mathbf{d}_{(k)} &= 0,
\end{aligned}$$

and we see that the search direction $\mathbf{d}_{(k)}$ satisfies (3.6). We now have the set of residuals $\mathbf{r}_{(0)}, \ldots, \mathbf{r}_{(k)}$ and the set of search directions $\mathbf{d}_{(0)}, \ldots, \mathbf{d}_{(k)}$ which satisfy (3.5) and (3.6) respectively. The next step is to show that $\mathbf{r}_{(k+1)}$ also belongs in the set of residuals satisfying (3.5).

In this case we also need a step stone to reach the goal. Consider for $i < k-1$

$$\begin{aligned}
\mathbf{d}_{(k)}{}^T\mathbf{A}\mathbf{d}_{(i)} &= (\mathbf{r}_{(k)} - \gamma_{(k)}\mathbf{d}_{(k-1)})^T\mathbf{A}\mathbf{d}_{(i)} \\
&= \mathbf{r}_{(k)}\mathbf{A}\mathbf{d}_{(i)} - \gamma_{(k)}\mathbf{d}_{(k-1)}{}^T\mathbf{A}\mathbf{d}_{(i)} \\
&= \mathbf{r}_{(k)}\mathbf{A}\mathbf{d}_{(i)}, \qquad (3.10)
\end{aligned}$$

which is used to show that the new residual is orthogonal to all previous residuals with the exception of the immediate previous, that is $i < k$ in the following

$$\begin{aligned}
\mathbf{r}_{(i)}^T\mathbf{r}_{(k+1)} &= \mathbf{r}_{(i)}^T(\mathbf{r}_{(k)} - \beta_{(k)}\mathbf{A}\mathbf{d}_{(k)}) \\
&= \mathbf{r}_{(i)}{}^T\mathbf{r}_{(k)} - \beta\mathbf{r}_{(i)}{}^T\mathbf{A}\mathbf{d}_{(k)} \\
&= 0 - \beta_{(k)}\mathbf{d}_{(i)}{}^T\mathbf{A}\mathbf{d}_{(k)} \\
&= 0.
\end{aligned}$$

It now remains to show that (3.5) also holds for $i = k$ — a task quite similar to the one performed in (3.7)

$$\begin{aligned}
\mathbf{r}_{(k)}{}^T\mathbf{r}_{(k+1)} &= \mathbf{r}_{(k)}{}^T(\mathbf{r}_{(k)} - \beta_{(k)}\mathbf{A}\mathbf{d}_{(k)}) \\
&= \mathbf{r}_{(k)}{}^T\mathbf{r}_{(k)} - \frac{\mathbf{r}_{(k)}{}^T\mathbf{r}_{(k)}}{\mathbf{d}_{(k)}{}^T\mathbf{A}\mathbf{d}_{(k)}}\mathbf{r}_{(k)}^T\mathbf{A}\mathbf{d}_{(k)} \\
&= \|\mathbf{r}_{(k)}\|^2 - \|\mathbf{r}_{(k)}\|^2\frac{\mathbf{r}_{(k)}^T\mathbf{A}\mathbf{d}_{(k)}}{\mathbf{d}_{(k)}^T\mathbf{A}\mathbf{d}_{(k)}} \\
&= \|\mathbf{r}_{(k)}\|^2 - \|\mathbf{r}_{(k)}\|^2\frac{\mathbf{d}_{(k)}^T\mathbf{A}\mathbf{d}_{(k)}}{\mathbf{d}_{(k)}^T\mathbf{A}\mathbf{d}_{(k)}} \\
&= 0,
\end{aligned}$$

and we can now add $\mathbf{r}_{(k+1)}$ to the set of residuals. This concludes the induction proof. □

Theorem 1 proves that CG terminates after *at most* n iterations because $\mathbf{r}_{(n)} = 0$. Due to this fact CG was at first viewed as a direct method. However due to the finite precision of real world computers CG looses the mutual orthogonality properties and it will not yield a residual of zero after n iterations. Because of this CG was "forgotten" until around 1970 CG was "rediscovered" as an iterative method. It has since been in the tool box for solving linear systems of equations [8], but the exact behaviour due to finite precision is still subject for investigation, see e.g. [9].

3.1.2 Solving Non-SPD Equation Systems

Conjugate gradients in its basic form only solves equation systems with an SPD coefficient matrix $\mathbf{A}$. To solve a non-symmetric or even non-square system $\mathbf{Kx} = \mathbf{y}$ of equations one can use the normal equations

$$\mathbf{K^T K x} = \mathbf{K^T y}. \tag{3.11}$$

If $\mathbf{K}$ has full column rank then $\mathbf{K}^T\mathbf{K}$ is SPD and CG can be applied without problems.

To solve (3.11) with CG one must take care during the calculations. An example is never to calculate the matrix $\mathbf{A} = \mathbf{K}^T\mathbf{K}$ because this leads to unnecessary inaccuracies. The procedure of solving the normal equations has many variations, but experience has shown the method denoted CGLS (Algorithm 3) to be the best choice in strong competition with the close cousin LSQR based on Lanczos bidiagonalization [24]. An actual implementations of CGLS (and LSQR) is available in REGULARIZATION TOOLS [17].

Algorithm 3 Least Square Conjugate Gradients (CGLS). Solve $\mathbf{K}^T\mathbf{Kx} = \mathbf{K}^T\mathbf{b}, \mathbf{K} \in \mathbb{R}^{n\times m},\ n > m$.

$\mathbf{r}_{(0)} = \mathbf{b} - \mathbf{Kx}_{(0)}$
$\mathbf{d}_{(0)} = \mathbf{K}^T\mathbf{r}_{(0)}$
$i \leftarrow 0$
while $\mathbf{r}_{(i)} \neq 0$ **do**
  $\beta_{(i)} = \frac{(\mathbf{Kr}_{(i)})^T\mathbf{Kr}_{(i)}}{(\mathbf{Kd}_{(i)})^T(\mathbf{Kd}_{(i)})}$
  $\mathbf{x}_{(i+1)} = \mathbf{x}_{(i)} + \beta_{(i)}\mathbf{d}_{(i)})$
  $\mathbf{r}_{(i+1)} = \mathbf{r}_{(i)} - \beta_{(i)}\mathbf{Kd}_{(i)}$
  $\gamma_{(i+1)} = \frac{(\mathbf{Kr}_{(i+1)})^T(\mathbf{Kr}_{(i+1)})}{(\mathbf{Kr}_{(i)})^T(\mathbf{Kr}_{(i)})}$
  $\mathbf{d}_{(i+1)} = \mathbf{K}^T\mathbf{r}_{(i+10)} + \gamma_{(i+1)}\mathbf{d}_{(i)}$
  $i \leftarrow i + 1$
end while

One downside to the normal equations is the squaring of the condition number, that is $\mathrm{cond}(\mathbf{K}^T\mathbf{K}) = \mathrm{cond}^2(\mathbf{K})$ (easily derived from the SVD of $\mathbf{K}$ and $\mathbf{K}^T\mathbf{K}$). The downside becomes apparent when we consider the convergence properties of CG — the subject of the next section.

3.2 Convergence Properties of CG

In this section we find expressions that characterize the convergence of CG and we discover that the distribution of the eigenvalues plays an important role for the speed of convergence. We transform the convergence problem to the well known problem of finding approximating polynomials — a problem with many useful results available, which then can be used in this, on first sight different, context.

In each step CG minimizes the square of the $\mathbf{A}$-norm of the error, that is

$$\|\mathbf{e}_{(i)}\|_{\mathbf{A}}^2 = \mathbf{e}_{(i)}^T\mathbf{A}\mathbf{e}_{(i)} \tag{3.12}$$

in the ith dimensional shifted Krylov subspace

$$\mathbf{e}_{(i)} \in \mathbf{e}_{(0)} + \mathcal{K}(\mathbf{A}, \mathbf{e}, i). \tag{3.13}$$

To establish this fact we first write $\mathbf{e}_{(0)}$ as a linear combination of the search directions

$$\mathbf{e}_{(0)} = \sum_{j=0}^{n-1} \delta_j \mathbf{d}_{(j)}.$$

The coefficients δ_i are well defined because a multiplication with $\mathbf{d}_{(k)}^T\mathbf{A}$ for $k = 0, \ldots, n-1$ yields

$$\begin{aligned}
\mathbf{d}_{(k)}^T\mathbf{A}\mathbf{e}_{(0)} &= \sum_{j=0}^{n-1} \delta_j \mathbf{d}_{(k)}^T\mathbf{A}\mathbf{d}_{(j)} \\
&= \delta_k \mathbf{d}_{(k)}^T\mathbf{A}\mathbf{d}_{(k)} \\
&\Downarrow \\
\delta_k &= \frac{\mathbf{d}_{(k)}^T\mathbf{A}\mathbf{e}_{(0)}}{\mathbf{d}_{(k)}^T\mathbf{A}\mathbf{d}_{(k)}}.
\end{aligned}$$

The error at step k is $\mathbf{e}_{(k)} = \mathbf{e}_{(0)} + \sum_{i=0}^{k-1} \beta_{(i)}\mathbf{d}_{(i)}$ and because the search directions are $\mathbf{A}$-orthogonal we get

$$\begin{aligned}
\delta_k &= \frac{\mathbf{d}_{(k)}^T\mathbf{A}\left(\mathbf{e}_{(0)} + \sum_{i=0}^{k-1} \beta_{(i)}\mathbf{d}_{(i)}\right)}{\mathbf{d}_{(k)}^T\mathbf{A}\mathbf{d}_{(k)}} \\
&= \frac{\mathbf{d}_{(k)}^T\mathbf{A}\mathbf{e}_{(k)}}{\mathbf{d}_{(k)}^T\mathbf{A}\mathbf{d}_{(k)}}.
\end{aligned}$$

Using (3.9) and that $\mathbf{e}_{(k)} = -\mathbf{A}\mathbf{r}_{(k)}$ we finally get

$$\begin{aligned}\delta_k &= -\frac{\mathbf{d}_{(k)}^T\mathbf{r}_{(k)}}{\mathbf{d}_{(k)}^T\mathbf{A}\mathbf{d}_{(k)}} \\ &= -\frac{\mathbf{r}_{(k)}^T\mathbf{r}_{(k)}}{\mathbf{d}_{(k)}^T\mathbf{A}\mathbf{d}_{(k)}} \\ &= -\beta_{(k)}. \end{aligned} \tag{3.14}$$

From (3.14) we get another view on CG, namely that at each iteration the error component in the current search direction is eliminated and that it will never reappear. The result also leads to the minimization property of $\|\mathbf{e}_{(k)}\|_{\mathbf{A}}^2$

$$\begin{aligned}\|\mathbf{e}_{(k)}\|_{\mathbf{A}}^2 &= \sum_{j=i}^{n-1}\sum_{k=i}^{n-1}\delta_j\delta_k\mathbf{d}_{(j)}^T\mathbf{A}\mathbf{d}_{(k)} \\ &= \sum_{j=1}^{n-1}\delta_j^2\mathbf{d}_{(j)}\mathbf{A}\mathbf{d}_{(j)}. \end{aligned} \tag{3.15}$$

We see that each term of this sum is associated with a search direction that has not yet been used. All other vectors in the shifted subspace (3.13) must have these components. Hence CG minimizes the $\mathbf{A}$-norm of the error in each step because it has no components besides the ones in (3.15).

The minimization property enables us to express the error in yet another form

$$\begin{aligned}\mathbf{e}_{(i)} &= \mathbf{e}_{(0)} + \eta_1\mathbf{A}\mathbf{e}_{(0)} + \ldots + \eta_i\mathbf{A}^i\mathbf{e}_{(0)} \\ &= \left(\mathbf{I} + \sum_{j=1}^{i}\eta_j\mathbf{A}^j\right)\mathbf{e}_{(0)}, \end{aligned} \tag{3.16}$$

where the coefficients η_j are chosen to minimize the error $\|\mathbf{e}_{(i)}\|_{\mathbf{A}}$. The term $(\mathbf{I} + \sum_{j=1}^{i}\eta_i\mathbf{A}^j)$ can be viewed as a polynomial in a broad sense. Let it be denoted $P_i(\cdot)$, where the argument can be either a matrix or a scalar. To clarify consider the polynomial $P_2(\cdot) = 1 + \cdot + 3\cdot^2$, which with a scalar argument yields $P_2(\lambda) = 1 + \lambda + 3\lambda^2$ while a matrix argument gives $\mathbf{P}_2(\mathbf{A}) = 1 + \mathbf{A} + 3\mathbf{A}^2$. If an eigenvalue pair $(\lambda, \mathbf{u})$ of the

matrix $\mathbf{A}$ is used we get, using the previous example, that $\mathbf{P}_i(\mathbf{A})\mathbf{u} = \mathbf{u} + \mathbf{A}\mathbf{u} + \mathbf{3}\mathbf{A}\mathbf{A}\mathbf{u} = \mathbf{u} + \lambda\mathbf{u} + 3\lambda^2\mathbf{u} = P_i(\lambda)\mathbf{u}$.

We now express the initial error $\mathbf{e}_{(0)}$ as a linear combination of $\mathbf{A}$'s normalized eigenvectors $\mathbf{u}_1, \ldots, \mathbf{u}_n$

$$\mathbf{e}_{(0)} = \sum_{j=1}^{n}\xi_j\mathbf{u}_j,$$

and in the same manner it is possible to write the ith error with the coefficients ξ_j and a polynomial of type (3.16)

$$\mathbf{e}_{(i)} = \sum_{j=1}^{n}\xi_j P_i(\lambda_j)\mathbf{u}_j. \tag{3.17}$$

From (3.17) we find an expression for $\|\mathbf{e}_{(i)}\|_{\mathbf{A}}^2$

$$\begin{aligned}\mathbf{A}\mathbf{e}_{(i)} &= \sum_{j=1}^{n}\xi_j P_i(\lambda_j)\lambda_j\mathbf{u}_j \\ &\Downarrow \\ \mathbf{e}_{(i)}^T\mathbf{A}\mathbf{e}_{(i)} &= \sum_{j=1}^{n}\xi_j^2 P_i^2(\lambda_j)\lambda_j \\ &= \|\mathbf{e}_{(i)}\|_{\mathbf{A}}^2, \end{aligned}$$

which we know CG minimizes. If we take $\Lambda(\mathbf{A})$ to be the set of eigenvalues of $\mathbf{A}$ we get due to the minimization property

$$\begin{aligned}\|\mathbf{e}_{(i)}\|_{\mathbf{A}}^2 &= \sum_{j=1}^{n}\xi_j^2 P_i^2(\lambda_j)\lambda_j \\ &\leq \min_{P_i}\max_{\lambda\in\Lambda(\mathbf{A})} P_i^2(\lambda)\sum_{j=1}^{n}\xi_j^2\lambda_j \\ &= \min_{P_i}\max_{\lambda\in\Lambda(\mathbf{A})} P_i^2(\lambda)\|\mathbf{e}_{(0)}\|_{\mathbf{A}}^2, \end{aligned} \tag{3.18}$$

where the non-indexed λ is the eigenvalue that maximizes $P_i^2(\lambda)$.

Equation (3.18) shows that the convergence problem can be transformed into a problem of picking the best polynomial of a degree equal to the iteration number, that approximates a set of points $\{(0,1)\} \cup \bigcup_{i=1}^{n}\{(\lambda_i, 0)\}$.

From the study of approximating polynomials we have the following results regarding picking the best polynomial and thereby also of importance to the convergence of CG:

- Having a set of N points $\{(x_i, y_i)|x_i \neq x_j\}$ a polynomial of degree N or less exists which has the property $P_N(x_i) = y_i$. This is another way of stating that CG terminates after at most N iterations. In fact CG terminates after a number of iterations equal to the number of distinct eigenvalues whose eigenvector is non-orthogonal to the error.
- Clustered eigenvalues makes it easier for a low degree polynomial to approximate zero in these points and we can expect CG to converge faster.
- If all eigenvalues are evenly distributed in the interval $[\lambda_{\min}; \lambda_{\max}]$ a reasonable approach is to minimize the maxima of the "absolute polynomial" $|P_i|$ on this interval. This is achieved by the the polynomial
$$P_i(\lambda) = \frac{T_i(\frac{\lambda_{\max}+\lambda_{\min}-2\lambda}{\lambda_{\max}-\lambda_{\min}})}{T_i(\frac{\lambda_{\max}+\lambda_{\min}}{\lambda_{\max}-\lambda_{\min}})},$$
where $T_i(\omega) = \frac{1}{2}((\omega + \sqrt{w^2-1})^i + (\omega - \sqrt{w^2-1})^i)$ is the Chebyshev polynomial (the proof is in [31]). Setting $(\text{cond}(\mathbf{A}) = \kappa)$ and using the fact that the numerator polynomial has a maximal value of 1 we get from (3.18) that
$$\begin{aligned} \|\mathbf{e}_{(i)}\|_{\mathbf{A}} &\leq T_i(\frac{\lambda_{\max}+\lambda_{\min}}{\lambda_{\max}-\lambda_{\min}})^{-1}\|\mathbf{e}_{(0)}\|_{\mathbf{A}} \\ &= T_i(\frac{\kappa+1}{\kappa-1})^{-1}\|\mathbf{e}_{(0)}\|_{\mathbf{A}} \\ &= 2\left(\left(\frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1}\right)^i + \left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^i\right)^{-1}\|\mathbf{e}_{(0)}\|_{\mathbf{A}}. \end{aligned}$$
The second part of the summation converges to zero as i increases and the expression is commonly simplified to
$$\|\mathbf{e}_{(i)}\|_{\mathbf{A}} \leq 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^i\|\mathbf{e}_{(0)}\|_{\mathbf{A}}. \tag{3.19}$$
From (3.19) we see that $\kappa \approx 1$ yields fast convergence but $\kappa \gg 1$ results in a possible slow convergence depending on whether the assumptions are true or not.

The statements just reviewed have shown true for real problems and they make a natural introduction to the concept of preconditioning. The finite precision problem results in slower convergence speed and a need for more iterations — but the rules are still applicable in most cases.

3.3 Preconditioning

In the previous section we saw how the convergence of CG depends on the the condition number and the eigenvalues of the coefficient matrix. Preconditioning is a method to improve convergence by lowering the condition number and/or increasing the eigenvalue clustering.

The idea is to solve the modified problem
$$\mathbf{N}^{-1}\mathbf{A}\mathbf{x} = \mathbf{N}^{-1}\mathbf{b}, \tag{3.20}$$
where $\mathbf{N}$ is a SPD matrix which is easy to invert. If $\text{cond}(\mathbf{N}^{-1}\mathbf{A}) < \text{cond}(\mathbf{A})$ or the eigenvalues of $\mathbf{N}^{-1}\mathbf{A}$ are more clustered than those of $\mathbf{A}$ we achieve a higher rate of convergence. A good preconditioner $\mathbf{N}$ can also be viewed as an approximation to $\mathbf{A}$ or, in other words, a good preconditioner is $\mathbf{N}^{-1} \approx \mathbf{A}^{-1}$ so that $\mathbf{N}^{-1}\mathbf{A}$ approximates the identity $\mathbf{I}$.

However, the matrix $\mathbf{N}^{-1}\mathbf{A}$ might not be SPD which is a necessity in the case of CG. The solution is to by solve the system
$$\mathbf{C}^{-1}\mathbf{A}\mathbf{C}^{-T}\hat{\mathbf{x}} = \mathbf{C}^{-1}\mathbf{b}, \qquad \hat{\mathbf{x}} = \mathbf{C}^T\mathbf{x},$$
where $\mathbf{C}$ is a factorization of $\mathbf{N} = \mathbf{C}\mathbf{C}^T$ (for example the Cholesky factorization or the "square root"). The system has the same solution as (3.20) and the coefficient matrix also have the same eigenvalues and eigenvectors as $\mathbf{N}^{-1}\mathbf{A}$. In order to prove this assume that $\mathbf{u}$ is an eigenvector of $\mathbf{N}^{-1}\mathbf{A}$ with eigenvalue λ. Then a simple calculation shows
$$\begin{aligned} (\mathbf{C}^{-1}\mathbf{A}\mathbf{C}^{-T})(\mathbf{C}^T\mathbf{u}) &= \mathbf{C}^{-1}\mathbf{A}\mathbf{u} \\ &= \mathbf{C}^T\mathbf{C}^{-T}\mathbf{C}^{-1}\mathbf{A}\mathbf{u} \\ &= \mathbf{C}^T(\mathbf{C}\mathbf{C}^T)^{-1}\mathbf{A}\mathbf{u} \\ &= \mathbf{C}^T\mathbf{N}^{-1}\mathbf{A}\mathbf{u} \\ &= \lambda\mathbf{C}^T\mathbf{u}. \end{aligned} \tag{3.21}$$

The preconditioned CG algorithm is derived by replacing the system matrix $\mathbf{A}$ with $\mathbf{C}^{-1}\mathbf{A}\mathbf{C}^{-T}$, $\mathbf{x}$ with $\mathbf{C}^T\mathbf{x}$, and $\mathbf{b}$ with $\mathbf{C}^{-1}\mathbf{b}$ in Algorithm 2, followed by substitutions of variables such that multiplications with $\mathbf{C}$ (and its transposed etc.) are avoided. The result of this tedious and boring work can be seen in Algorithm 4. In fact we do not need to form the preconditioning matrix $\mathbf{N}$ explicitly if we are able to implicitly apply $\mathbf{N}^{-1}$ to some vector. If calculations are reused the cost of preconditioning amounts to one solution with the preconditioner $\mathbf{N}$.

Algorithm 4 Preconditioned Conjugate Gradients (PCG). Solve $\mathbf{A}\mathbf{x} = \mathbf{b}$, $\mathbf{A}, \mathbf{N} \in \mathbb{R}^{n \times n}$ where both $\mathbf{A}$ and $\mathbf{N}$ are SPD.

$\mathbf{r}_{(0)} = \mathbf{b} - \mathbf{A}\mathbf{x}_{(0)}$
$\mathbf{d}_{(0)} = \mathbf{N}^{-1}\mathbf{b}$ % Preconditioning step
$i \leftarrow 0$
while $\mathbf{r}_{(i)} \neq \mathbf{0}$ **do**
  $\beta_{(i)} = \frac{\mathbf{r}_{(i)}^T \mathbf{N}^{-1} \mathbf{r}_{(i)}}{\mathbf{d}_{(i)}^T \mathbf{A} \mathbf{d}_{(i)}}$ % Preconditioning step
  $\mathbf{x}_{(i+1)} = \mathbf{x}_{(i)} + \beta_{(i)} \mathbf{d}_{(i)}$
  $\mathbf{r}_{(i+1)} = \mathbf{r}_{(i)} - \beta_{(i)} \mathbf{A} \mathbf{d}_{(i)}$
  $\gamma_{(i+1)} = \frac{\mathbf{r}_{(i+1)}^T \mathbf{N}^{-1} \mathbf{r}_{(i+1)}}{\mathbf{r}_{(i)}^T \mathbf{N}^{-1} \mathbf{r}_{(i)}}$
  $\mathbf{d}_{(i+1)} = \mathbf{N}^{-1}\mathbf{r}_{(i+1)} + \gamma_{(i+1)} \mathbf{d}_{(i)}$
  $i \leftarrow i + 1$
end while

3.3.1 Well known preconditioners

Many preconditioners have been tested and used since the revival of CG around 1970. Some are iterative methods like the Jacobi and Gauss-Seidel while other take another approach. A list of the best known preconditioners are found in [2] but we will only list a few that have relevance to this thesis:

- Jacobi preconditioning uses the diagonal of $\mathbf{A}$ and has been shown useful if the diagonal elements are relatively different.
- Gauss-Seidel preconditioning is originally an iterative method itself that sequentially satisfies one equation from the system at the time and then proceeds to the next. Gauss-Seidel has a symmetric variant which is suitable for preconditioning.

- Incomplete factorization preconditioning uses an approximation to $\mathbf{A}$ which is easy to invert. The idea is to do a fast but inaccurate factorization of $\mathbf{A}$. An example could be only to calculate elements of an Cholesky factor in already non zero places.
- Block versions of all of the above.

Unfortunately none of these are particularly suited for ill-posed problems and more specialized techniques must be used. They all try to improve on the hole spectrum of eigenvalues, but in the context of ill-posed problems only the large eigenvalues hold our interest — while the small eigenvalues are unwanted. However, in the next chapter we learn that both the Jacobi and the Gauss-Seidel preconditioner in their block versions have given inspiration to the two-grid preconditioners that concentrate their efforts on the large eigenvalues.

Chapter 4

Ill-Posed Problems and Conjugate Gradients

Several symmetric systems, some involving as many as twelve unknowns, have been solved on the IBM card programmed calculator. In one case, where the ratio of the largest to the smallest eigenvalue was 4.9, a satisfactory solution has been obtained already in the third step; in another case, where this ratio was 100, one had to carry out fifteen steps in order to get an estimate with six correct digits. From [18].

The previous chapter focused on the CG algorithm in a general setting. This chapter takes CG into the land of ill-posed problems. We will first experience the regularizing effect of CG but the main subject will be the development of two preconditioners and a variation of CG which are specifically tailored towards ill-posed problems and their special properties. Theory on the expected condition number is also included.

4.1 Regularization and Conjugate Gradients

The convergence of CG depends, as we have already seen, on the condition number of the coefficient matrix. This leaves us with little hope when dealing with ill-posed problems where we face a huge condition number. However CG still has some nice properties. At each step CG minimizes the error in the Krylov subspace $\mathcal{K}(\mathbf{A}, \mathbf{b}, k)$ and if $\mathcal{K}(\mathbf{A}, \mathbf{b}, k)$ for example

approximates the k-dimensional subspace spanned by the first k right singular vectors then CG will yield a result close to the TSVD solution with truncation parameter k.

Hansen, O'Leary and Stewart show in [16, Theorem 6.4.2] that the first k filter factors from the kth iteration of CG, i.e. $f_i^{(k)},\ i = 1 \ldots k$, are close to 1 — or more precisely that $|f_i^{(k)} - 1|$ is bounded — while the last $n - k$ filter factors converge to zero[1]. This result shows that limiting the iteration number will give a regularization effect, but eventually CG converges to the unwanted least square solution (all $f_i = 1$). This "convergence" behaviour is called semiconvergence.

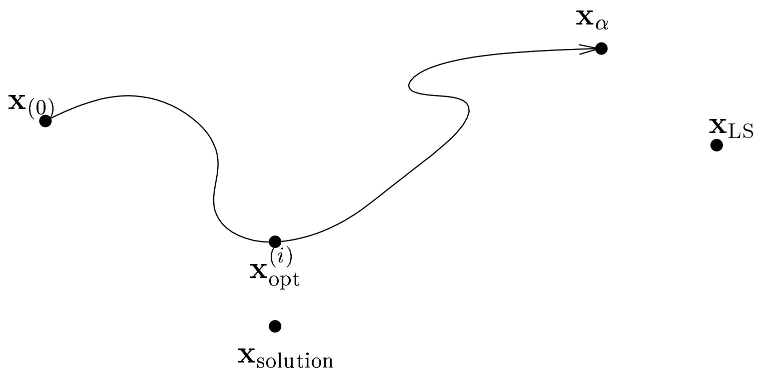


Figure 4.1: *Illustration of semiconvergence. The iterations follow the line from $\mathbf{x}_{(0)}$ to the Tikhonov solution $\mathbf{x}_\alpha$. Initially the iterations converge towards the solution $\mathbf{x}_{\mathrm{solution}}$ but from $\mathbf{x}_{\mathrm{opt}}^{(i)}$ it starts to diverge from the solution as it approaches the least square solution. If the system had been unregularized we would see convergence towards $\mathbf{x}_{\mathrm{LS}}$ instead.*

Another view on CG and regularization is to apply CG to the Tikhonov regularized system ($\mathbf{L} = \mathbf{I}_n$ for simplicity)

$$(\mathbf{K}^T\mathbf{K} + \alpha\mathbf{I}_n)\mathbf{x} = \mathbf{K}^T\mathbf{y}. \tag{4.1}$$

The Tikhonov system has a smaller condition number than the ordinary least squares system $\mathbf{K}^T\mathbf{K}$ because the eigenvalues of $\mathbf{K}^T\mathbf{K} + \alpha\mathbf{I}_n$ relate

[1] The result is derived for the infinite precision CG.

to the SVD of $\mathbf{K}^T\mathbf{K} = \mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^T$ as follows

$$\begin{aligned}(\mathbf{K}^T\mathbf{K} + \alpha\mathbf{I}_n) &= \mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^T + \alpha\mathbf{I}_n \\ &= \mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^T + \alpha\mathbf{V}\mathbf{I}_n\mathbf{V}^T \\ &= \mathbf{V}(\mathbf{\Sigma}^2 + \alpha\mathbf{I}_n)\mathbf{V}^T. \end{aligned} \tag{4.2}$$

From (4.2) we see that the condition number of the Tikhonov regularized equation system for $\alpha > \sigma_n^2$ is

$$\mathrm{cond}(\mathbf{K}^T\mathbf{K} + \alpha\mathbf{I}_n) = \frac{\sigma_1^2 + \alpha}{\sigma_n^2 + \alpha} \le 1 + \frac{\sigma_1^2}{\alpha} \approx \frac{\sigma_1^2}{\alpha}.$$

Despite the improvement of the condition number a preconditioner could very well be necessary in order to solve the system within a reasonable number of iterations.

Finally it must be mentioned that the two regularization methods do not exclude each other, so one can use the formulation (4.1) combined with a limitation on the number of iterations. This idea is illustrated in Figure 4.1.

4.2 Two-Grid Methods

Hanke and Vogel proposed in [13] a type of preconditioners specifically tailored for ill-posed problems. In [12] they further examines and test their idea. Riley has in his Ph.D. thesis [29] investigated the computational costs of the algorithms presented in the following. The multilevel idea is based on an article by Rieder [28] which in turn is inspired by an article by King [20].

We are going to look at two preconditioners and a variation of CG called Schur complement CG. They all rely on a subspace splitting which gave Hanke and Vogel inspiration to denote the methods two-level. However, in this thesis the methods are denoted two-grid because we believe that the term two-grid describes the structure of the algorithms better. The next many pages are based on [13], [12] and [29] but the derivations take smaller steps and explanations are more plentiful in order to enhance "quick learning".

The methods are aimed at a symmetric, positive definite system

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{4.3}$$

where $\mathbf{A} \in \mathbb{R}^{n\times n}$ and $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$ comes from the Tikhonov regularized normal equation introduced in Section 2.1

$$\mathbf{A} = \mathbf{K}^T\mathbf{K} + \alpha\mathbf{L}^T\mathbf{L}\ , \quad \mathbf{b} = \mathbf{K}^T\mathbf{y}.$$

Choosing $\mathbf{L} = \mathbf{I}_n$ gives some nicer/shorter formulas but they are naturally more specialized and less general — unless a reformulation of the problem is used [16, page 38ff]. We will derive the algorithms for a general $\mathbf{L}$ but in this section we assume that $\mathbf{L}^T\mathbf{L}$ is SPD. Section 4.3 covers the complementary case where $\mathbf{L}^T\mathbf{L}$ is semidefinite, i.e., $\mathbf{L}^T\mathbf{L}$ has a nontrivial null space. To shorten and simplify the notation we set $\mathbf{M} = \mathbf{L}^T\mathbf{L} \in \mathbb{R}^{n\times n}$.

4.2.1 A Two-Grid Splitting

The two levels/grids are formed by a splitting of the space $\mathbb{R}^{n\times n}$ into a k-dimensional subspace $\mathcal{V}_k$ and a $(n-k)$-dimensional subspace $\mathcal{W}_k$ where $\mathcal{V}_k$ and $\mathcal{W}_k$ together spans $\mathbb{R}^{n\times n}$. The subspaces must be $\mathbf{M}$-orthogonal, i.e., a vector $\mathbf{v} \in \mathbb{R}^n$ has a unique representation $\mathbf{u} = \mathbf{v} + \mathbf{w}$ where $\mathbf{v} \in \mathcal{V}_k$, $\mathbf{w} \in \mathcal{W}_k$ and $\mathbf{v}^T\mathbf{M}\mathbf{w} = 0$. Let the columns of $\mathbf{\Phi} \in \mathbb{R}^{n\times k}$ form a basis of $\mathcal{V}_k$ and let the columns of $\mathbf{\Psi} \in \mathbb{R}^{n\times n-k}$ form a $\mathbf{M}$ orthonormal basis for $\mathcal{W}_k$. The requirements above tell us that

$$\mathbf{\Phi}^T\mathbf{M}\mathbf{\Psi} = \mathbf{0}_{k,n-k}, \tag{4.4}$$
$$\mathbf{\Psi}^T\mathbf{M}\mathbf{\Psi} = \mathbf{I}_{n-k}. \tag{4.5}$$

The subspace $\mathcal{V}_k$ is in [13] denoted the coarse subspace for reasons explained later. Think of $\mathbf{\Phi} = [\boldsymbol{\phi}_1\ \boldsymbol{\phi}_2\ \dots\ \boldsymbol{\phi}_k]$ as an operator from $\mathbb{R}^n$ into $\mathbb{R}^k$ and $\mathbf{\Psi} = [\boldsymbol{\psi}_1\ \boldsymbol{\psi}_2\ \dots\ \boldsymbol{\psi}_{n-k}]$ as an operator from $\mathbb{R}^n$ into $\mathbb{R}^{(n-k)}$. To accompany $\mathbf{\Phi}$ and $\mathbf{\Psi}$ we define the matrices

$$\mathbf{G} = \mathbf{\Phi}^T\mathbf{M}\mathbf{\Phi}, \tag{4.6}$$
$$\mathbf{P} = \mathbf{\Phi}\mathbf{G}^{-1}\mathbf{\Phi}^T\mathbf{M}, \tag{4.7}$$
$$\mathbf{Q} = \mathbf{\Psi}\mathbf{\Psi}^T\mathbf{M} = \mathbf{I}_n - \mathbf{P}, \tag{4.8}$$

where $\mathbf{G} \in \mathbb{R}^{k\times k}$ and $\mathbf{P}, \mathbf{Q} \in \mathbb{R}^{n\times n}$.

Observe that the inverse of $\mathbf{G}$ is defined because $\mathbf{M}$ is assumed to be SPD and hence invertible. We now go through a couple of proofs on the properties of $\mathbf{P}$ and $\mathbf{Q}$. Steps from the proof are used in later derivations and in a sense it is not the theorem itself that is important but rather the proof.

Theorem 2 *The matrix $\mathbf{P}$ is an $\mathbf{M}$-orthogonal projector onto $\mathcal{V}_k$ and $\mathbf{Q}$ is the complementary $\mathbf{M}$-orthogonal projector onto $\mathcal{W}_k$.*

Proof. Simple calculations show that $\mathbf{P}$ is a projection matrix

$$\begin{aligned}\mathbf{PP} &= \mathbf{\Phi G}^{-1}\mathbf{\Phi}^T\mathbf{M\Phi G}^{-1}\mathbf{\Phi}^T\mathbf{M}\\ &= \mathbf{\Phi G}^{-1}\mathbf{GG}^{-1}\mathbf{\Phi}^T\mathbf{M}\\ &= \mathbf{\Phi G}^{-1}\mathbf{\Phi}^T\mathbf{M}\\ &= \mathbf{P}\end{aligned} \tag{4.9}$$

and due to (4.5) $\mathbf{Q}$ is also a projector

$$\begin{aligned}\mathbf{QQ} &= \mathbf{\Psi\Psi}^T\mathbf{M\Psi\Psi}^T\mathbf{M}\\ &= \mathbf{\Psi I}_{n-k}\mathbf{\Psi}^T\mathbf{M}\\ &= \mathbf{\Psi\Psi}^T\mathbf{M}\\ &= \mathbf{Q}.\end{aligned} \tag{4.10}$$

The projectors are $\mathbf{M}$-orthogonal because of (4.4)

$$\begin{aligned}\mathbf{P}^T\mathbf{MQ} &= (\mathbf{\Phi G}^{-1}\mathbf{\Phi}^T\mathbf{M})^T\mathbf{M\Psi\Psi}^T\mathbf{M}\\ &= \mathbf{M\Phi G}^{-T}\mathbf{\Phi}^T\mathbf{M\Psi\Psi}^T\mathbf{M}\\ &= \mathbf{M\Phi G}^{-T}\mathbf{0}_{k,n-k}\mathbf{\Psi}^T\mathbf{M}\\ &= \mathbf{0}.\end{aligned}$$

Furthermore we see that

$$\begin{aligned}\mathbf{P\Phi} &= \mathbf{\Phi G}^{-1}\mathbf{\Phi}^T\mathbf{M\Phi}\\ &= \mathbf{\Phi G}^{-1}\mathbf{G}\\ &= \mathbf{\Phi}\end{aligned}$$

and (using (4.5))

$$\begin{aligned}\mathbf{Q\Psi} &= \mathbf{\Psi\Psi}^T\mathbf{M\Psi}\\ &= \mathbf{\Psi}.\end{aligned} \tag{4.11}$$

Any vector $\mathbf{u} \in \mathbb{R}^n$ has an unique representation $\mathbf{u} = \mathbf{\Phi a} + \mathbf{\Psi b}$, where $\mathbf{a} \in \mathbb{R}^k$ and $\mathbf{b} \in \mathbb{R}^{n-k}$. From

$$\begin{aligned}(\mathbf{P}+\mathbf{Q})\mathbf{u} &= \mathbf{P\Phi a} + \mathbf{P\Psi b} + \mathbf{Q\Phi a} + \mathbf{Q\Psi b}\\ &= \mathbf{\Phi a} + \mathbf{\Phi G}^{-1}\mathbf{\Phi}^T\mathbf{M\Psi b} + \mathbf{\Psi\Psi}^T\mathbf{M\Phi a} + \mathbf{\Psi b}\\ &= \mathbf{u},\end{aligned}$$

we see that $\mathbf{P} + \mathbf{Q} = \mathbf{I}_n$, i.e., the projection matrices project into complementary subspaces. □

The next step is to split the equation system (4.3) into a block matrix system. Adjoining $\mathbf{\Phi}$ and $\mathbf{\Psi}$ into a full rank matrix $[\mathbf{\Phi}\ \mathbf{\Psi}] \in \mathbb{R}^{n\times n}$ enables the transformation

$$[\mathbf{\Phi}\ \mathbf{\Psi}]^T\mathbf{A}[\mathbf{\Phi}\ \mathbf{\Psi}][\mathbf{\Phi}\ \mathbf{\Psi}]^{-1}\mathbf{x} = [\mathbf{\Phi}\ \mathbf{\Psi}]^T\mathbf{b}. \tag{4.12}$$

Defining

$$\mathbf{u} = \begin{bmatrix}\mathbf{u}_1\\ \mathbf{u}_2\end{bmatrix} = [\mathbf{\Phi}\ \mathbf{\Psi}]^{-1}\mathbf{x} \quad \Leftrightarrow \quad \mathbf{x} = \mathbf{\Phi u}_1 + \mathbf{\Psi u}_2 \tag{4.13}$$

and splitting the system into blocks we get

$$\begin{bmatrix}\mathbf{\Phi}^T\mathbf{A\Phi} & \mathbf{\Phi}^T\mathbf{A\Psi}\\ \mathbf{\Psi}^T\mathbf{A\Phi} & \mathbf{\Psi}^T\mathbf{A\Psi}\end{bmatrix}\begin{bmatrix}\mathbf{u}_1\\ \mathbf{u}_2\end{bmatrix} = \begin{bmatrix}\mathbf{\Phi}^T\mathbf{b}\\ \mathbf{\Psi}^T\mathbf{b}\end{bmatrix}. \tag{4.14}$$

Introducing the notation

$$\mathbf{A}_{11} = \mathbf{\Phi}^T\mathbf{A\Phi} \in \mathbb{R}^{k\times k}, \tag{4.15}$$
$$\mathbf{A}_{12} = \mathbf{\Phi}^T\mathbf{A\Psi} \in \mathbb{R}^{k\times n-k}, \tag{4.16}$$
$$\mathbf{A}_{21} = \mathbf{\Psi}^T\mathbf{A\Phi} \in \mathbb{R}^{n-k\times k}, \tag{4.17}$$
$$\mathbf{A}_{22} = \mathbf{\Psi}^T\mathbf{A\Psi} \in \mathbb{R}^{n-k\times n-k}, \tag{4.18}$$

where the index indicates the position within the block matrix simplifies the expression a bit;

$$\begin{bmatrix}\mathbf{A}_{11} & \mathbf{A}_{12}\\ \mathbf{A}_{21} & \mathbf{A}_{22}\end{bmatrix}\begin{bmatrix}\mathbf{u}_1\\ \mathbf{u}_2\end{bmatrix} = \begin{bmatrix}\mathbf{\Phi}^T\mathbf{b}\\ \mathbf{\Psi}^T\mathbf{b}\end{bmatrix}. \tag{4.19}$$

The constraints (4.4) and (4.5) on the columns of $\mathbf{\Phi}$ and $\mathbf{\Psi}$ simplify (4.14) to

$$\begin{bmatrix}\mathbf{\Phi}^T\mathbf{A\Phi} & \mathbf{\Phi}^T\mathbf{K}^T\mathbf{K\Psi}\\ \mathbf{\Psi}^T\mathbf{K}^T\mathbf{K\Phi} & \mathbf{\Psi}^T\mathbf{K}^T\mathbf{K\Psi} + \alpha\mathbf{I}_{n-k}\end{bmatrix}\begin{bmatrix}\mathbf{u}_1\\ \mathbf{u}_2\end{bmatrix} = \begin{bmatrix}\mathbf{\Phi}^T\mathbf{b}\\ \mathbf{\Psi}^T\mathbf{b}\end{bmatrix}. \tag{4.20}$$

The basic idea of the Jacobi and symmetric Gauss-Seidel inspired preconditioners described later is to discard $\mathbf{\Psi}^T\mathbf{K}^T\mathbf{K}\mathbf{\Psi}$ and leave $\alpha\mathbf{I}_{n-k}$ from the the bottom right part of the coefficient matrix ($\mathbf{A}_{22}$) and use this as an approximation to $[\mathbf{\Phi}\ \ \mathbf{\Psi}]^T\mathbf{A}[\mathbf{\Phi}\ \ \mathbf{\Psi}]$. To justify this deletion consider the the SVD decomposition of $\mathbf{K} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$. Then let $\mathbf{\Psi} = \mathbf{V}_{k+1:n} = [\mathbf{v}_{k+1}, \ldots, \mathbf{v}_n]$ be the last $n-k$ right singular vectors and we see that

$$\begin{aligned}
\mathbf{\Psi}^T\mathbf{K}^T\mathbf{K}\mathbf{\Psi} + \alpha\mathbf{I}_{n-k} &= \mathbf{\Psi}^T\mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^T\mathbf{\Psi} + \alpha\mathbf{I}_{n-k} \\
&= \mathbf{V}^T_{k+1:n}[\mathbf{V}_{1:k}\,\mathbf{V}_{k+1:n}]\mathbf{\Sigma}^2\begin{bmatrix}\mathbf{V}^T_{1:k}\\ \mathbf{V}^T_{k+1:n}\end{bmatrix}\mathbf{V}_{k+1:n} + \alpha\mathbf{I}_{n-k} \\
&= [\mathbf{0}_{n-k,k}\ \mathbf{I}_{n-k}]\,\mathbf{\Sigma}^2\begin{bmatrix}\mathbf{0}_{n-k,k}\\ \mathbf{I}_{k-n}\end{bmatrix} + \alpha\mathbf{I}_{n-k} \\
&= \mathbf{\Sigma}^2_{k+1:n,k+1:n} + \alpha\mathbf{I}_{n-k}.
\end{aligned}$$

If α is chosen such that $\alpha > \sigma^2_{k+1}$, we see that the term $\alpha\mathbf{I}_{n-k}$ "dominates" $\mathbf{W}^T\mathbf{K}^T\mathbf{K}\mathbf{W}$ — in other words $\mathbf{A}_{22} = \mathbf{W}^T\mathbf{K}^T\mathbf{K}\mathbf{W} + \alpha\mathbf{I}_{n-k} \approx \alpha\mathbf{I}_{n-k}$.

This leads to an equation system with the coefficient matrix

$$\widehat{\mathbf{A}} = \begin{bmatrix}\mathbf{\Phi}^T\mathbf{A}\mathbf{\Phi} & \mathbf{\Phi}^T\mathbf{K}^T\mathbf{K}\mathbf{\Psi}\\ \mathbf{\Psi}^T\mathbf{K}^T\mathbf{K}\mathbf{\Phi} & \alpha\mathbf{I}_{n-k}\end{bmatrix} = \begin{bmatrix}\mathbf{A}_{11} & \mathbf{A}_{12}\\ \mathbf{A}_{21} & \widehat{\mathbf{A}}_{22}\end{bmatrix}, \tag{4.21}$$

which we will split into three matrices for later use ($\widehat{\mathbf{A}} = \mathbf{L} + \widehat{\mathbf{D}} + \mathbf{U}$):

$$\widehat{\mathbf{D}} = \begin{bmatrix}\mathbf{A}_{11} & \mathbf{0}\\ \mathbf{0} & \widehat{\mathbf{A}}_{22}\end{bmatrix},\quad \mathbf{L} = \begin{bmatrix}\mathbf{0} & \mathbf{0}\\ \mathbf{A}_{21} & \mathbf{0}\end{bmatrix},\quad \mathbf{U} = \begin{bmatrix}\mathbf{0} & \mathbf{A}_{12}\\ \mathbf{0} & \mathbf{0}\end{bmatrix}. \tag{4.22}$$

4.2.2 Jacobi Preconditioning

The standard Jacobi preconditioner is formed by the diagonal (or block diagonal) of the coefficient matrix. If we take the formulation from (4.21) and delete the off diagonal blocks we get a Jacobi-like preconditioner which in [13], [12] and others are called the "Additive Schwarz Preconditioner"

$$\mathbf{N}_J = \widehat{\mathbf{D}} = \begin{bmatrix}\mathbf{\Phi}^T\mathbf{A}\mathbf{\Phi} & \mathbf{0}_{k,n-k}\\ \mathbf{0}_{n-k,k} & \alpha\mathbf{I}_{n-k}\end{bmatrix}. \tag{4.23}$$

It is possible to use (4.23) directly in a standard preconditioned conjugate gradient method like the one in Algorithm 4. However, the residual and

iteration vector must be transformed according to (4.12). Hence the total preconditioning step takes the form

$$\mathbf{x} = [\mathbf{\Phi}\ \mathbf{\Psi}]\mathbf{N}_J^{-1}[\mathbf{\Phi}\ \mathbf{\Psi}]^T\mathbf{r}. \tag{4.24}$$

We will now show how to eliminate the explicit use of $\mathbf{\Psi}$ and how otherwise to exploit the structure of the preconditioner. When using (4.23) as a preconditioner it is necessary to solve systems of the type

$$\begin{aligned}
\mathbf{N}_J\mathbf{u} &= [\mathbf{\Phi}\ \mathbf{\Psi}]^T\,\mathbf{r} \\
&\Updownarrow \\
\mathbf{A}_{11}\mathbf{u}_1 &= \mathbf{\Phi}^T\mathbf{r} \qquad (4.25)\\
\alpha\mathbf{u}_2 &= \mathbf{\Psi}^T\mathbf{r}. \qquad (4.26)
\end{aligned}$$

Starting out from equation (4.26) and using the results from (4.9)-(4.11) we get

$$\begin{aligned}
\mathbf{\Psi}\mathbf{u}_2 &= \frac{1}{\alpha}\mathbf{\Psi}\mathbf{\Psi}^T\mathbf{r} \\
&= \frac{1}{\alpha}\mathbf{\Psi}\mathbf{\Psi}^T\mathbf{M}\mathbf{M}^{-1}\mathbf{r} \\
&= \frac{1}{\alpha}\mathbf{Q}\mathbf{M}^{-1}\mathbf{r} \\
&= \frac{1}{\alpha}(\mathbf{I} - \mathbf{V}\mathbf{G}^{-1}\mathbf{\Phi}^T\mathbf{M})\mathbf{M}^{-1}\mathbf{r} \\
&= \frac{1}{\alpha}(\mathbf{M}^{-1} - \mathbf{\Phi}\mathbf{G}^{-1}\mathbf{\Phi}^T)\mathbf{r}. \qquad (4.27)
\end{aligned}$$

Finally we use inversion of $\mathbf{A}_{11}$ to solve (4.25), and insertion into (4.13) yields

$$\begin{aligned}
\mathbf{x} &= \mathbf{\Phi}\mathbf{u}_1 + \mathbf{\Psi}\mathbf{x}_2 \\
&= \mathbf{\Phi}(\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{r}) + \frac{1}{\alpha}\mathbf{M}^{-1}\mathbf{r} - \frac{1}{\alpha}\mathbf{\Phi}\mathbf{G}^{-1}\mathbf{\Phi}^T\mathbf{r} \\
&= \mathbf{\Phi}(\mathbf{A}_{11}^{-1} + \frac{1}{\alpha}\mathbf{G}^{-1})\mathbf{\Phi}^T\mathbf{r} + \frac{1}{\alpha}\mathbf{M}^{-1}\mathbf{r}, \qquad (4.28)
\end{aligned}$$

and we have avoided use explicit use of $\mathbf{\Psi}$. This formula can be plugged directly into the preconditioned CG (Algorithm 4). However because the preconditioning step takes place at each iteration an initial factorization of $\mathbf{A}_{11}$ is preferable. A Cholesky factorization is a good choice because $\mathbf{A}_{11}$ is SPD. Similar considerations can be applied to the inversion of $\mathbf{M}$, but this case calls for a QR factorization because a QR factorization of $\mathbf{L} = \mathbf{QR}$ also carries a factorization of $\mathbf{M} = \mathbf{L}^T\mathbf{L} = \mathbf{R}^T\mathbf{Q}^T\mathbf{QR} = \mathbf{R}^T\mathbf{R}$. A Matlab program using the Jacobi-like two-grid preconditioner is listed in appendix A.3.1.

We note that the preconditioning step involves solving one system with $\mathbf{A}_{11}$, one with $\mathbf{G}$ and one system with $\mathbf{M}$. The matrices $\mathbf{A}_{11}, \mathbf{G} \in \mathbb{R}^{k\times k}$ both depend on the dimension of $\mathcal{V}_k$. Finally we have 3 applications of $\mathbf{\Phi}$ or $\mathbf{\Phi}^T$ and the use of $\mathbf{A}$ in the actual CG iteration. The solution of a system with $\mathbf{M} \in \mathbb{R}^{n\times n}$ looks problematic at first, but in most applications $\mathbf{M}$ is banded or has some other structure which makes the factorization and solution fast.

The Connection to Multigrid

We have previously denoted $\mathcal{V}_k$ the "coarse subspace" and we have called the preconditioner two-grid. The term coarse subspace originates from multigrid methods [3]. A fundamental multigrid operation can be described as a three step process:

1. A vector (the current result) is projected onto a coarse grid — a restriction.
2. The system is solved (exactly) on the coarse subspace (easy because it is smaller). A correction is hereby obtained.
3. The correction is projected back onto the original (fine) grid — a prolongation — where it is used to correct the fine solution.

Equation (4.23) includes the part $\mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{r}$ which can be interpreted as a transformation onto a coarse grid (restriction), an exact solution and finally a projection back (prolongation). Hence we have a two-grid method.

The obvious step from a two-grid to a three-grid method has been attempted by Vogel [35] but the three-grid preconditioner showed *worse* results than the much more simple two-grid strategy. We will in this thesis only treat and discuss two-grid methods because of Vogel's results.

4.2.3 Symmetric Gauss-Seidel Preconditioning

In contrast to the Jacobi-like method the symmetric Gauss-Seidel-like preconditioner also involves the off diagonal blocks. A matrix factorization of the preconditioner takes the form

$$\mathbf{N}_{GS} = (\widehat{\mathbf{D}} + \mathbf{L})\widehat{\mathbf{D}}^{-1}(\widehat{\mathbf{D}} + \mathbf{U}). \tag{4.29}$$

Note that the Jacobi and Gauss-Seidel two-grid preconditioners are equal if the subspaces $\mathcal{V}_k$ and $\mathcal{W}_k$ are constructed from the columns of the matrix $\mathbf{V}$ of the SVD (in the case $\mathbf{L} \neq \mathbf{I}$ it would be the columns of the matrix usually denoted $\mathbf{X}$ from the GSVD). This choice yields zero off-diagonal parts, i.e., $\mathbf{L} = \mathbf{U}^T = \mathbf{0}_{n-k,k}$, and diagonal diagonal blocks.

As in the case of the Jacobi two-grid preconditioner it is possible to reformulate the solution of $\mathbf{N}_{GS}\mathbf{u} = [\mathbf{\Phi}\ \mathbf{\Psi}]^T\mathbf{r}$ such that the actual creation of $\mathbf{\Psi}$ can be avoided. Maybe even more important we avoid $\mathbf{A}_{12}$ and its transposed $\mathbf{A}_{21}$ that are large and most likely full.

First we split $\mathbf{r}$

$$[\mathbf{\Phi}\ \mathbf{\Psi}]^T\,\mathbf{r} = \begin{bmatrix} \mathbf{\Phi}^T\mathbf{r} \\ \mathbf{\Psi}^T\mathbf{r} \end{bmatrix} = \begin{bmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 \end{bmatrix} \tag{4.30}$$

Premultiplying $\mathbf{r}$ with $\mathbf{N}_{GS}^{-1} = (\mathbf{D}+\mathbf{U})^{-1}\mathbf{D}(\mathbf{D}+\mathbf{L})^{-1}$ yields in three steps

$$\begin{aligned}
(\widehat{\mathbf{D}} + \mathbf{L})^{-1}\mathbf{r} &= \begin{bmatrix} \mathbf{A}_{11}^{-1}\mathbf{r}_1 \\ \widehat{\mathbf{A}}_{22}^{-1}(\mathbf{r}_2 - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{r}_1) \end{bmatrix}, \\
\mathbf{D}(\mathbf{D}+\mathbf{L})^{-1}\mathbf{r} &= \begin{bmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{r}_1 \end{bmatrix}, \\
(\mathbf{D}+\mathbf{U})^{-1}\mathbf{D}(\mathbf{D}+\mathbf{L})^{-1}\mathbf{r} &= \begin{bmatrix} \mathbf{A}_{11}^{-1}(\mathbf{r}_1 - \mathbf{A}_{12}\widehat{\mathbf{A}}_{22}^{-1}(\mathbf{r}_2 - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{r}_1)) \\ \widehat{\mathbf{A}}_{22}^{-1}(\mathbf{r}_2 - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{r}_1) \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix}.
\end{aligned} \tag{4.31}$$

A substitution of the definitions of $\mathbf{A}_{11}$, $\mathbf{r}_1$ etc. from (4.12) and (4.30) into (4.31) produces the following expressions for $\mathbf{u}_1$ and $\mathbf{u}_2$

$$\mathbf{u}_1 = \mathbf{A}_{11}^{-1}(\mathbf{\Phi}^T\mathbf{r} - \mathbf{\Phi}^T\mathbf{K}^T\mathbf{K}\mathbf{\Psi}\frac{1}{\alpha}(\mathbf{\Psi}^T\mathbf{r} - \mathbf{\Psi}^T\mathbf{K}^T\mathbf{K}\mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{r})) \tag{4.32}$$

$$\mathbf{u}_2 = \frac{1}{\alpha}(\mathbf{\Psi}^T\mathbf{r} - \mathbf{\Psi}^T\mathbf{K}^T\mathbf{K}\mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{r}). \tag{4.33}$$

The solution of the preconditioning is given by $\mathbf{x} = \mathbf{\Phi u}_1 + \mathbf{\Psi u}_2$ and in order to simplify the expression we utilize the equations (4.9)-(4.11) and obtain

$$\begin{aligned}
\mathbf{\Psi u}_2 &= \frac{1}{\alpha}\mathbf{\Psi\Psi}^T(\mathbf{r} - \mathbf{K}^T\mathbf{K\Phi A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{r}) \\
&= \frac{1}{\alpha}\mathbf{\Psi\Psi}^T\mathbf{MM}^{-1}(\mathbf{r} - \mathbf{K}^T\mathbf{K\Phi A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{r}) \\
&= \frac{1}{\alpha}(\mathbf{I}_n - \mathbf{P})\mathbf{M}^{-1}(\mathbf{r} - \mathbf{K}^T\mathbf{K\Phi A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{r}) \\
&= \frac{1}{\alpha}(\mathbf{M}^{-1} - \mathbf{\Phi G}^{-1}\mathbf{\Phi}^T)(\mathbf{r} - \mathbf{K}^T\mathbf{K\Phi A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{r}) \qquad (4.34)
\end{aligned}$$

and

$$\begin{aligned}
\mathbf{\Phi u}_1 &= \mathbf{\Phi A}_{11}^{-1}(\mathbf{\Phi}^T\mathbf{r} - \mathbf{\Phi}^T\mathbf{K}^T\mathbf{K\Psi}\frac{1}{\alpha}(\mathbf{\Psi}^T\mathbf{r} - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{r})) \\
&= \mathbf{\Phi A}_{11}^{-1}\mathbf{\Phi}^T(\mathbf{r} - \mathbf{K}^T\mathbf{K\Psi u}_2). \qquad (4.35)
\end{aligned}$$

The formulas have several possibilities for reuse of already calculated quantities, for example (4.34) should be evaluated first and the result is then used in (4.35) whereby the use of $\mathbf{\Psi}$ is avoided even though it appears in (4.35). This is used in Algorithm 5. The algorithm lays the ground for the Matlab implementation in Appendix A.3.2.

Algorithm 5 Gauss-Seidel two-grid preconditioning, $\mathbf{x} = \mathbf{N}_{GS}^{-1}\mathbf{r}$, for SPD $\mathbf{L}^T\mathbf{L}$. The comments indicate the equation from which the calculation is extracted.

1: $\mathbf{v} = \mathbf{\Phi A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{r}$ % Eq. (4.34)
2: $\mathbf{e} = \mathbf{r} - \mathbf{K}^T\mathbf{Kv}$ % Eq. (4.34)
3: $\mathbf{w} = \mathbf{M}^{-1}\mathbf{e}$ % Eq. (4.34)
4: $\mathbf{u} = \mathbf{w} - \mathbf{\Phi G}^{-1}\mathbf{\Phi}^T\mathbf{e}$ % Eq. (4.34)
5: $\mathbf{x}_W = \frac{1}{\alpha}\mathbf{u}$ % Eq. (4.34)
6: $\mathbf{y} = \mathbf{r} - \mathbf{K}^T\mathbf{Kx}_W$ % Eq. (4.35)
7: $\mathbf{x}_V = \mathbf{\Phi A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{y}$ % Eq. (4.35)
8: $\mathbf{x} = \mathbf{x}_V + \mathbf{x}_W$

Scanning Algorithm 5 we count two solutions with coefficient matrix $\mathbf{A}_{11}$, one solution with $\mathbf{G}$ and one solution with $\mathbf{M}$. The algorithm uses two applications of $\mathbf{K}^T\mathbf{K}$ and 6 applications of $\mathbf{\Phi}$ or $\mathbf{\Phi}^T$. To this the the actual CG step must be added implying one application of $\mathbf{A}$.

The Connection to Domain Decomposition

It is known from the literature, e.g. [32], that Jacobi and Gauss-Seidel in their block versions are similar to the additive and multiplicative Schwarz methods respectively. The sources of our methods [13], [12] and [29] also denote the preconditioners as Schwarz methods and the similarities are undoubtedly many.

The actual algorithms are equal if $\mathbf{A}_{22}$ is unchanged. Furthermore the subspaces $\mathcal{V}_k$ and $\mathcal{W}_k$ must be created from the columns of the identity matrix according to the domain decomposition, i.e., $[1\ 0\ \ldots\ 0]^T$ would be a column in $\mathbf{\Phi}$ if "point 1" is in domain 1 and if it is in domain 2 it is placed in $\mathbf{\Psi}$. However, domain decomposition usually operates with *overlapping* regions/domains where we, for our methods, have separate domains and no overlap in the sense that $\mathcal{V}_k$ and $\mathcal{W}_k$ are $\mathbf{M}$-orthogonal.

4.2.4 Schur Complement Conjugate Gradients

Both the Jacobi and the symmetric Gauss-Seidel preconditioner assume $\|\mathbf{\Psi}^T\mathbf{K}^T\mathbf{K\Psi}\|_2$ to be much smaller than α and hence they disregard it. Schur complement CG takes another approach and does not discard the $\mathbf{\Psi}^T\mathbf{K}^T\mathbf{K\Psi}$ part. As the name indicates it is not a preconditioner but a variant of CG.

Applying block Gaussian elimination to (4.20) we get

$$\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{0} & \mathbf{A}_{22} - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{A}_{12} \end{bmatrix} \begin{bmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{\Phi}^T\mathbf{b} \\ \mathbf{\Psi}^T\mathbf{b} - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{b} \end{bmatrix},$$

where the lower right block,

$$\mathbf{S} = \mathbf{A}_{22} - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{A}_{12} = \mathbf{\Psi}^T\mathbf{A}(\mathbf{I} - \mathbf{\Phi A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{A})\mathbf{\Psi}, \qquad (4.36)$$

is denoted the Schur complement to $\mathbf{A}_{11}$ in the coefficient matrix (4.20).

Isolating $\mathbf{u}_1$ in the first equation yields

$$\mathbf{u}_1 = \mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{b} - \mathbf{A}_{11}^{-1}\mathbf{A}_{12}\mathbf{u}_2, \qquad (4.37)$$

and rearranging the second equation gives

$$\mathbf{Su}_2 = \mathbf{\Psi}^T\mathbf{b} - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{b}. \qquad (4.38)$$

We see that if we solve (4.38) with respect to $\mathbf{u}_2$ we obtain $\mathbf{u}_1$ easily using (4.37).

The Schur complement $\mathbf{S} \in \mathbb{R}^{(n-k)\times(n-k)}$ is SPD because $\mathbf{A}$ is SPD [1, Theorem 3.9] and furthermore $\mathbf{S}$ is large (we would like k to be small because of the factorizations of $\mathbf{A}_{11}$ and $\mathbf{G}$). Hence CG comes to mind as a reasonable choice for solving (4.38). We would like the algorithm to work without explicit knowledge of $\mathbf{\Psi}$ and the off diagonal blocks $\mathbf{A}_{12}$ and $\mathbf{A}_{21}$ because of their size. Furthermore we would like the algorithm to work on vectors in terms of the original system such that the iteration vector and residual is available during the iterations. This is a bit more complicated than we have seen for Jacobi and Gauss-Seidel preconditioning and the derivation is longer and more complicated.

Recall (Algorithm 2) that CG needs to compute the following three quantities

$$\begin{aligned} \widetilde{\mathbf{r}} &= \widetilde{\mathbf{b}} - \mathbf{S}\widetilde{\mathbf{x}}, && (4.39)\\ \widetilde{\rho} &= \widetilde{\mathbf{r}}^T\widetilde{\mathbf{r}}, && (4.40)\\ \widetilde{\beta} &= \frac{\widetilde{\rho}}{\widetilde{\mathbf{d}}^T\mathbf{S}\widetilde{\mathbf{d}}}, && (4.41) \end{aligned}$$

where the vectors with a tilde denotes vectors in the "Schur complement space", e.g. we have $\widetilde{\mathbf{x}} = \mathbf{u}_2$.

For a start we need a suitable starting guess which we obtain by looking at the expression $\mathbf{x} = \mathbf{\Phi}\mathbf{u}_1 + \mathbf{\Psi}\mathbf{u}_2$,

$$\begin{aligned} \mathbf{x} &= \mathbf{\Phi}(\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{b} - \mathbf{A}_{11}^{-1}\mathbf{A}_{12}\mathbf{u}_2) + \mathbf{\Psi}\mathbf{u}_2 \\ &= \mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{b} - \mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{A}\mathbf{\Psi}\mathbf{u}_2 + \mathbf{\Psi}\mathbf{u}_2 \\ &= \mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{b} + (\mathbf{I}_n - \mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{A})\mathbf{\Psi}\mathbf{u}_2. && (4.42) \end{aligned}$$

Equation (4.42) hints that the starting guess $\mathbf{x}_{(0)} = \mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{b}$ would be appropriate. Then the results, that is the iteration vectors, are simple updates of the starting guess $\mathbf{x}_{(i)} = \mathbf{x}_{(0)} + (\mathbf{I}_n - \mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{A})\mathbf{\Psi}\mathbf{u}_2^{(i)}$. We omit the iteration index hereafter to unclutter the text and formulas.

Next we find an expression for the residual

$$\begin{aligned} \mathbf{r} &= \mathbf{b} - \mathbf{A}\mathbf{x} \\ &= \mathbf{b} - \mathbf{A}\mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{b} + \mathbf{A}(\mathbf{I} - \mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{A})\mathbf{\Psi}\mathbf{u}_2 \\ &= (\mathbf{I} - \mathbf{A}\mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T)(\mathbf{b} - \mathbf{A}\mathbf{\Psi}\mathbf{u}_2), \end{aligned}$$

and for the residual in the Schur complement space $\widetilde{\mathbf{r}}$ we get

$$\begin{aligned} \widetilde{\mathbf{r}} &= \widetilde{\mathbf{b}} - \mathbf{S}\mathbf{u}_2 \\ &= (\mathbf{\Psi}^T\mathbf{b} - \mathbf{\Psi}^T\mathbf{A}\mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{b}) - (\mathbf{\Psi}^T\mathbf{A}\mathbf{\Psi} - \mathbf{\Psi}^T\mathbf{A}\mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{A}\mathbf{\Psi})\mathbf{u}_2 \\ &= \mathbf{\Psi}^T(\mathbf{b} - \mathbf{A}\mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{b} - \mathbf{A}\mathbf{\Psi}\mathbf{u}_2 + \mathbf{A}\mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{A}\mathbf{\Psi}\mathbf{u}_2) \\ &= \mathbf{\Psi}^T((\mathbf{I} - \mathbf{A}\mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T)(\mathbf{b} - \mathbf{A}\mathbf{\Psi}\mathbf{u}_2)) \\ &= \mathbf{\Psi}^T\mathbf{r}, && (4.43) \end{aligned}$$

which enables us to find $\widetilde{\rho}$ by

$$\begin{aligned} \widetilde{\rho} &= \widetilde{\mathbf{r}}^T\widetilde{\mathbf{r}} \\ &= \mathbf{r}^T\mathbf{\Psi}\mathbf{\Psi}^T\mathbf{r} \\ &= \mathbf{r}^T\mathbf{\Psi}\mathbf{\Psi}^T\mathbf{M}\mathbf{M}^{-1}\mathbf{r} \\ &= \mathbf{r}^T\mathbf{Q}\mathbf{M}^{-1}\mathbf{r} \\ &= \mathbf{r}^T(\mathbf{M}^{-1} - \mathbf{\Phi}\mathbf{G}^{-1}\mathbf{\Phi}^T)\mathbf{r}. && (4.44) \end{aligned}$$

This equation can be further reduced owing to

$$\begin{aligned} \mathbf{\Phi}^T\mathbf{r} &= \mathbf{\Phi}^T(\mathbf{I} - \mathbf{A}\mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T)(\mathbf{b} - \mathbf{A}\mathbf{\Psi}\mathbf{u}_2) \\ &= (\mathbf{\Phi}^T - \mathbf{\Phi}^T\mathbf{A}\mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T)(\mathbf{b} - \mathbf{A}\mathbf{\Psi}\mathbf{u}_2) \\ &= (\mathbf{\Phi}^T - \mathbf{A}_{11}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T)(\mathbf{b} - \mathbf{A}\mathbf{\Psi}\mathbf{u}_2) \\ &= \mathbf{0}_{k,l}, \end{aligned}$$

and thus we have

$$\widetilde{\rho} = \mathbf{r}^T\mathbf{M}^{-1}\mathbf{r}. \qquad (4.45)$$

Equation (4.43) suggests that we define the decent direction $\widetilde{\mathbf{d}}$ by

$$\widetilde{\mathbf{d}} = \mathbf{\Psi}^T\mathbf{d}\,, \quad \mathbf{d} \in \mathbb{R}^n$$

and applying the Schur complement yields

$$\begin{aligned}
\mathbf{S}\widetilde{\mathbf{d}} &= \mathbf{\Psi}^T\mathbf{A}(\mathbf{I}_n - \mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{A})\mathbf{\Psi}\mathbf{\Psi}^T\mathbf{d}\\
&= \mathbf{\Psi}^T\mathbf{A}(\mathbf{I}_n - \mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{A})\mathbf{\Psi}\mathbf{\Psi}^T\mathbf{M}\mathbf{M}^{-1}\mathbf{d}\\
&= \mathbf{\Psi}^T\mathbf{A}(\mathbf{I}_n - \mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{A})(\mathbf{I}_n - \mathbf{\Phi}\mathbf{G}^{-1}\mathbf{\Phi}^T\mathbf{M})\mathbf{M}^{-1}\mathbf{d}\\
&= \mathbf{\Psi}^T\mathbf{A}(\mathbf{I}_n - \mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{A} - (\mathbf{I}_n - \mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{A})(\mathbf{\Phi}\mathbf{G}^{-1}\mathbf{\Phi}^T\mathbf{M}))\mathbf{M}^{-1}\mathbf{d}\\
&= \mathbf{\Psi}^T\mathbf{A}(\mathbf{I}_n - \mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{A} - (\mathbf{\Phi} - \mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{A}\mathbf{\Phi})(\mathbf{G}^{-1}\mathbf{\Phi}^T\mathbf{M}))\mathbf{M}^{-1}\mathbf{d}\\
&= \mathbf{\Psi}^T\mathbf{A}(\mathbf{I}_n - \mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{A} - (\mathbf{\Phi} - \mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{A}_{11})(\mathbf{G}^{-1}\mathbf{\Phi}^T\mathbf{M}))\mathbf{M}^{-1}\mathbf{d}\\
&= \mathbf{\Psi}^T\mathbf{A}(\mathbf{I}_n - \mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{A} - (\mathbf{0}_{n,k})(\mathbf{G}^{-1}\mathbf{\Phi}^T\mathbf{M}))\mathbf{M}^{-1}\mathbf{d}\\
&= \mathbf{\Psi}^T\mathbf{A}(\mathbf{I}_n - \mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{A})\mathbf{M}^{-1}\mathbf{d}.
\end{aligned} \tag{4.46}$$

For any $\widetilde{\mathbf{d}} \in \mathbb{R}^{n-k}$ define $\mathbf{w} \in \mathbb{R}^n$ by

$$\mathbf{w} = \mathbf{A}(\mathbf{I}_n - \mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{A})\mathbf{M}^{-1}\mathbf{d} \tag{4.47}$$

and by the definition of the Schur complement (4.36) we simplify (4.46) to

$$\mathbf{S}\widetilde{\mathbf{d}} = \mathbf{\Psi}^T\mathbf{w}.$$

It is now straightforward but tedious to calculate the denominator of $\widetilde{\beta} = \widetilde{\rho}/(\widetilde{\mathbf{d}}^T\mathbf{S}\widetilde{\mathbf{d}})$

$$\begin{aligned}
\widetilde{\mathbf{d}}^T\mathbf{S}\widetilde{\mathbf{d}} &= \mathbf{d}^T\mathbf{\Psi}\mathbf{\Psi}^T\mathbf{w}\\
&= \mathbf{d}^T\mathbf{\Psi}\mathbf{\Psi}^T\mathbf{M}\mathbf{M}^{-1}\mathbf{w}\\
&= \mathbf{d}^T(\mathbf{M}^{-1} - \mathbf{\Phi}\mathbf{G}^{-1}\mathbf{\Phi}^T)\mathbf{w}\\
&= \mathbf{d}^T\mathbf{M}^{-1}\mathbf{w} - \mathbf{d}^T\mathbf{\Phi}\mathbf{G}^{-1}\mathbf{\Phi}^T\mathbf{A}(\mathbf{I}_n - \mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{A})\mathbf{M}^{-1}\mathbf{d}\\
&= \mathbf{d}^T\mathbf{M}^{-1}\mathbf{w} - \mathbf{d}^T\mathbf{\Phi}\mathbf{G}^{-1}(\mathbf{\Phi}^T\mathbf{A} - \mathbf{\Phi}^T\mathbf{A}\mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{A})\mathbf{M}^{-1}\mathbf{d}\\
&= \mathbf{d}^T\mathbf{M}^{-1}\mathbf{w} - \mathbf{d}^T\mathbf{\Phi}\mathbf{G}^{-1}(\mathbf{\Phi}^T\mathbf{A} - \mathbf{A}_{11}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{A})\mathbf{M}^{-1}\mathbf{d}\\
&= \mathbf{d}^T\mathbf{M}^{-1}\mathbf{w} - \mathbf{d}^T\mathbf{\Phi}\mathbf{G}^{-1}(\mathbf{0}_{k,n})\mathbf{M}^{-1}\mathbf{d}\\
&= \mathbf{d}^T\mathbf{M}^{-1}\mathbf{w}.
\end{aligned} \tag{4.48}$$

Finally the update of the solution is done by

$$\begin{aligned}
\mathbf{x}_{(i+1)} &= \mathbf{x}_{(i)} + \beta_{(i)}(\mathbf{I}_n - \mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{A})\mathbf{\Psi}\widetilde{\mathbf{d}}_{(i)}\\
&= \mathbf{x}_{(i)} + \beta_{(i)}(\mathbf{I}_n - \mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{A})\mathbf{\Psi}\mathbf{\Psi}^T\mathbf{d}_{(i)}\\
&= \mathbf{x}_{(i)} + \beta_{(i)}(\mathbf{I}_n - \mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{A})\mathbf{\Psi}\mathbf{\Psi}^T\mathbf{M}\mathbf{M}^{-1}\mathbf{d}_{(i)}\\
&= \mathbf{x}_{(i)} + \beta_{(i)}(\mathbf{I}_n - \mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{A})\mathbf{M}^{-1}\mathbf{d}_{(i)},
\end{aligned} \tag{4.49}$$

where the last step is similar to (4.46). The original CG algorithm uses iterative updates of the residual and the search direction which of course also apply to Schur CG. In addition we implement the following updates

$$\begin{aligned}
\mathbf{y}_{(i)} &= \mathbf{M}^{-1}\mathbf{r}_{(i)}\\
&\Downarrow\\
\mathbf{y}_{(i+1)} &= \mathbf{M}^{-1}\mathbf{r}_{(i+1)}\\
&= \mathbf{M}^{-1}\mathbf{r}_{(i)} - \beta_{(i)}\mathbf{M}^{-1}\mathbf{w}_{(i)}\\
&= \mathbf{y}_{(i)} - \beta_{(i)}\mathbf{M}^{-1}\mathbf{w}_{(i)},
\end{aligned} \tag{4.50}$$

and

$$\begin{aligned}
\mathbf{z}_{(i)} &= \mathbf{M}^{-1}\mathbf{d}_{(i)}\\
&\Downarrow\\
\mathbf{z}_{(i+1)} &= \mathbf{M}^{-1}\mathbf{d}_{(i+1)}\\
&= \mathbf{M}^{-1}(\mathbf{r}_{(i+1)} + \gamma_{(i+1)}\mathbf{d}_{(i)})\\
&= \mathbf{y}_{(i+1)} + \gamma_{(k+1)}\mathbf{z}_{(i)}.
\end{aligned} \tag{4.51}$$

Algorithm 6 shows the use of the just presented formulas and an actual implementation is listed in appendix A.3.3.

Finally we sum up the costs of Algorithm 6. We see that each iteration involves solving one system with $\mathbf{A}_{11}$ and one with $\mathbf{M}$. In addition we find two applications of $\mathbf{A}$ and two applications of $\mathbf{\Phi}$ or $\mathbf{\Phi}^T$.

4.3 Two-Grid Methods part II, Semidefinite $\mathbf{L}^T\mathbf{L}$

The previously presented algorithms assume $\mathbf{L}^T\mathbf{L}$ to be SPD, but this is not always preferable. An example is the discrete representation of the

Algorithm 6 Schur complement CG. Solves $\mathbf{A}\mathbf{x} = \mathbf{b}$. The matrix $\mathbf{M} = \mathbf{L}^T\mathbf{L}$ must be SPD and hence invertible. Comments show the originating equation.

$\mathbf{x}_{(0)} = \mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{b}$ % Eq. (4.42)
$\mathbf{r}_{(0)} = \mathbf{b} - \mathbf{A}\mathbf{x}_{(0)}$
$\mathbf{y}_{(0)} = \mathbf{M}^{-1}\mathbf{r}_{(0)}$ % Eq. (4.45)
$\mathbf{d}_{(0)} = \mathbf{r}_{(0)}$
$\mathbf{z}_{(0)} = \mathbf{y}_{(0)}$
$\rho_{(0)} = \mathbf{y}_{(0)}^T\mathbf{r}_{(0)}$
$i \leftarrow 0$
repeat
  $\mathbf{v}_{(i)} = (\mathbf{I} - \mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{A})\mathbf{z}_{(i)}$ % Eq. (4.47)
  $\mathbf{w}_{(i)} = \mathbf{A}\mathbf{v}_{(i)}$ % Eq. (4.47)
  $\mathbf{g}_{(i)} = \mathbf{M}^{-1}\mathbf{w}_{(i)}$ % Eq. (4.48)
  $\beta_{(i)} = \rho_{(i)}/\mathbf{d}_{(i)}^T\mathbf{g}_{(i)}$ % Eq. (4.48)
  $\mathbf{x}_{(i+1)} = \mathbf{x}_{(i)} + \beta_{(i)}\mathbf{v}_{(i)}$ % Eq. (4.49)
  $\mathbf{r}_{(i+1)} = \mathbf{r}_{(i)} - \beta_{(i)}\mathbf{w}_{(i)}$
  $\mathbf{y}_{(i+1)} = \mathbf{y}_{(i)} - \beta_{(i)}\mathbf{g}_{(i)}$ % Eq. (4.50)
  $\rho_{(i+1)} = \mathbf{y}_{(i+1)}^T\mathbf{r}_{(i+1)}$ % Eq. (4.45)
  $\gamma_{(i+1)} = \rho_{(i+1)}/\rho_{(i)}$
  $\mathbf{d}_{(i+1)} = \mathbf{r}_{(i+1)} + \gamma_{(i+1)}\mathbf{d}_{(i)}$
  $\mathbf{z}_{(i+1)} = \mathbf{y}_{(i+1)} + \gamma_{(i+1)}\mathbf{z}_{(i)}$ % Eq. (4.51)
  $i \leftarrow i + 1$
until stop

first derivative which has a null space spanned by $[1\ 1 \ldots 1]^T$. Thus we face troubles when we wish to invert $\mathbf{L}^T\mathbf{L}$ and $\mathbf{G}$. Three solutions to this problem are available:

1. A fairly common fix is to add boundary conditions to $\mathbf{L}$, e.g. homogeneous Neumann boundary conditions, and thereby changing $\mathbf{L}$ to be invertible.
2. Modify the problem into "standard form" [16, Section 2.3.1]. This technique absorbs $\mathbf{L}$ into $\mathbf{A}$ and thereafter uses the identity as the penalty operator. The identity is SPD and the already treated algorithms can be applied.
3. Modify the algorithms such that they take the null space of $\mathbf{L}$ into consideration. One such approach is the implicit transformation [16, Section 3.2.2] used in the preconditioned iterative methods from REGULARIZATION TOOLS [17].

All three approaches are usable but we will only deal with the third and adapt the algorithms to the nontrivial null space of $\mathbf{L}^T\mathbf{L}$.

The modifications take place already in the creation of the subspaces $\mathcal{V}_k$ and $\mathcal{W}_k$ on which we now enforce further restrictions. First we need the l-dimensional null space of $\mathbf{L}$ to be a subspace of $\mathcal{V}_k$, that is $\mathcal{N}(\mathbf{L}) \subset \mathcal{V}_k$. The null space of $\mathbf{L}$ is normally given from the choice of $\mathbf{L}$, e.g. the second derivative has the its null space spanned by $\{[1\ 1\ \ldots\ 1]^T,\ [1\ 2\ \ldots\ n]^T\}$. Let the first l columns of $\mathbf{\Phi}$ be an orthogonal basis of $\mathcal{N}(\mathbf{L})$ with unit length and force the the last $k-l$ columns to be orthogonal to the first l columns. Furthermore make the columns of $\mathbf{\Psi}$ orthogonal to the null space of $\mathbf{L}$ (remember that we only use $\mathbf{\Psi}$ implicitly and this restriction is therefore only used during the derivations). Thus the matrix $\mathbf{\Phi} \in \mathbb{R}^{n\times k}$ has two parts

$$\mathbf{\Phi} = [\mathbf{\Phi}_0\ \mathbf{\Phi}_\perp] \quad \text{where} \mathbf{\Phi}_0 \in \mathbb{R}^{n\times l},\ \mathbf{\Phi}_\perp \in \mathbb{R}^{n\times k-l}.$$

Following the same pattern as with an invertible and SPD regularization matrix $\mathbf{M} = \mathbf{L}^T\mathbf{L}$ we define the matrices

$$\begin{aligned}
\mathbf{G} &= \mathbf{\Phi}_\perp^T\mathbf{M}\mathbf{\Phi}_\perp, && (4.52)\\
\mathbf{P}_\perp &= \mathbf{\Phi}_\perp\mathbf{G}^{-1}\mathbf{\Phi}_\perp^T\mathbf{M}, && (4.53)\\
\mathbf{P}_0 &= \mathbf{\Phi}_0\mathbf{\Phi}_0^T, && (4.54)\\
\mathbf{P} &= \mathbf{P}_0 + \mathbf{P}_\perp, && (4.55)\\
\mathbf{Q} &= \mathbf{\Psi}\mathbf{\Psi}^T\mathbf{M} = \mathbf{I} - \mathbf{P}, && (4.56)
\end{aligned}$$

where the columns of $\mathbf{\Psi}$ once again span $\mathcal{W}_k$. The matrix $\mathbf{G}$ is invertible because $\mathbf{\Phi}_\perp$ in (4.52) takes any vector into the orthogonal complement of $\mathcal{N}(\mathbf{M})$. Now a we go through a couple of derivations similar to those made in the SPD case but taking the null space of $\mathbf{M}$ into account.

Theorem 3 *Assume $\mathcal{N}(L) = \mathcal{R}(\mathbf{\Phi}_0)$. If the bases for for $\mathcal{V}_k$ and $\mathcal{W}_k$ defined by the columns $\mathbf{\Phi}$ and $\mathbf{\Psi}$ respectively satisfy (4.4) and (4.5) then the operators $\mathbf{P}$ and $\mathbf{Q}$ are $\mathbf{M}$-orthogonal projection operators onto $\mathcal{V}_k$ and $\mathcal{W}_k$ respectively.*

Proof The proofs that $\mathbf{Q} = \mathbf{Q}^2$ and $\mathbf{Q\Psi} = \mathbf{\Psi}$ are equal to the SPD case, see (4.10) and (4.11). For $\mathbf{Q\Phi}$ we get

$$\begin{aligned}\mathbf{Q\Phi} &= \mathbf{Q}[\mathbf{\Phi}_0\ \mathbf{\Phi}_\perp] \\ &= [\mathbf{0}_{n,k}\ \mathbf{0}_{n,n-k}] \\ &= \mathbf{0}_n.\end{aligned}$$

We see for $\mathbf{P}_\perp$ that

$$\begin{aligned}\mathbf{P}_\perp\mathbf{P}_\perp &= \mathbf{\Phi}_\perp\mathbf{G}^{-1}\mathbf{\Phi}_\perp^T\mathbf{M}\mathbf{\Phi}_\perp\mathbf{G}^{-1}\mathbf{\Phi}_\perp^T\mathbf{M} \\ &= \mathbf{\Phi}_\perp\mathbf{G}^{-1}\mathbf{G}\mathbf{G}^{-1}\mathbf{\Phi}_\perp^T\mathbf{M} \\ &= \mathbf{\Phi}_\perp\mathbf{G}^{-1}\mathbf{\Phi}_\perp^T\mathbf{M} \\ &= \mathbf{P}_\perp\end{aligned}$$

and in the case of $\mathbf{P}_0$

$$\begin{aligned}\mathbf{P}_0\mathbf{P}_0 &= \mathbf{\Phi}_0\mathbf{\Phi}_0^T\mathbf{\Phi}_0\mathbf{\Phi}_0^T \\ &= \mathbf{\Phi}_0\mathbf{I}_l\mathbf{\Phi}_0^T \\ &= \mathbf{P}_0.\end{aligned}$$

Because the columns of $\mathbf{\Phi}_0$ are orthonormal to the columns of $\mathbf{\Phi}_\perp$ we have that

$$\begin{aligned}\mathbf{P}_0\mathbf{P}_\perp &= \mathbf{\Phi}_0\mathbf{\Phi}_0^T\mathbf{\Phi}_\perp\mathbf{G}^{-1}\mathbf{\Phi}_\perp^T\mathbf{M} \\ &= \mathbf{\Phi}_0\mathbf{0}_{l,k-l}\mathbf{G}^{-1}\mathbf{\Phi}_\perp^T\mathbf{M} \\ &= \mathbf{0}_n\end{aligned}$$

and because $\mathbf{\Phi}_0$ spans $\mathcal{N}(\mathbf{M})$ we get

$$\begin{aligned}\mathbf{P}_\perp\mathbf{P}_0 &= \mathbf{\Phi}_\perp\mathbf{G}^{-1}\mathbf{\Phi}_\perp^T\mathbf{M}\mathbf{\Phi}_0\mathbf{\Phi}_0^T \\ &= \mathbf{\Phi}_\perp\mathbf{G}^{-1}\mathbf{\Phi}_\perp^T\mathbf{0}_{n,l}\mathbf{\Phi}_0^T \\ &= \mathbf{0}_n.\end{aligned}$$

Now we calculate

$$\begin{aligned}\mathbf{PP} &= (\mathbf{P}_0 + \mathbf{P}_\perp)(\mathbf{P}_0 + \mathbf{P}_\perp) \\ &= \mathbf{P}_0\mathbf{P}_0 + \mathbf{P}_0\mathbf{P}_\perp + \mathbf{P}_\perp\mathbf{P}_0 + \mathbf{P}_\perp\mathbf{P}_\perp \\ &= \mathbf{P}_0 + \mathbf{0}_n + \mathbf{0}_n + \mathbf{P}_\perp \\ &= \mathbf{P}\end{aligned}$$

and combined with the fact that

$$\begin{aligned}\mathbf{P\Psi} &= \mathbf{P}_0\mathbf{\Psi} + \mathbf{P}_\perp\mathbf{\Psi} \\ &= \mathbf{0}_n\end{aligned}$$

we conclude that $\mathbf{P}$ and $\mathbf{Q}$ are projection operators with $\mathbf{Q}$ projecting into $\mathcal{W}_k$. To establish that $\mathbf{P}$ projects into $\mathcal{V}_k$ note that any $\mathbf{v} \in \mathcal{V}_k$ can be written as a linear combination of the form $\mathbf{v} = a_1\phi_1 + a_1\phi_2 + \cdots + a_k\phi_k = \mathbf{\Phi a}$. If we multiply $\mathbf{v}$ with $\mathbf{P}$ we get

$$\begin{aligned}\mathbf{Pv} &= (\mathbf{P}_0 + \mathbf{P}_\perp)\mathbf{\Phi a} \\ &= (\mathbf{P}_0 + \mathbf{P}_\perp)[\mathbf{\Phi}_0\ \mathbf{\Phi}_\perp]\mathbf{a} \\ &= [(\mathbf{P}_0 + \mathbf{P}_\perp)\mathbf{\Phi}_0,\ (\mathbf{P}_0 + \mathbf{P}_\perp)\mathbf{\Phi}_\perp]\,\mathbf{a} \\ &= [\mathbf{\Phi}_0\ \mathbf{\Phi}_\perp]\mathbf{a} \\ &= \mathbf{\Phi a} = \mathbf{v}\end{aligned}$$

and we see that $\mathbf{P}$ is a projection operator into $\mathcal{V}_k$ □

With these results in mind we are now ready to extend the two-grid preconditioners to the case of an symmetric and semidefinite $\mathbf{L}^T\mathbf{L}$.

Jacobi Preconditioning Revisited

From the derivation of the Jacobi two-grid preconditioner we have the expression

$$\mathbf{N}_J^{-1}\mathbf{r} = \mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{r} + \frac{1}{\alpha}\mathbf{\Psi}\mathbf{\Psi}^T\mathbf{r}, \tag{4.57}$$

which we by means of the operators presented in (4.52)-(4.56) rewrite such that the explicit use of $\mathbf{\Psi}$ is avoided. For the derivation we need the pseudo inverse of $\mathbf{M}$ denoted by $\mathbf{M}^\dagger$. From the definition of the pseudo inverse (2.8) it is clear that $\mathbf{MM}^\dagger$ forms an orthogonal projection operator onto the complement of the null space of $\mathbf{M}$. Because the columns of $\boldsymbol{\Psi}$ and $\boldsymbol{\Phi}_\perp$ are orthogonal to the same null space we have

$$\begin{aligned}\boldsymbol{\Psi}^T\mathbf{M}\mathbf{M}^\dagger &= \boldsymbol{\Psi}^T, &(4.58)\\ \boldsymbol{\Phi}_\perp^T\mathbf{M}\mathbf{M}^\dagger &= \boldsymbol{\Phi}_\perp^T, &(4.59)\\ \boldsymbol{\Phi}_0^T\mathbf{M}\mathbf{M}^\dagger &= \mathbf{0}_{l,n}. &(4.60)\end{aligned}$$

Simple calculations then show

$$\begin{aligned}\boldsymbol{\Psi}\boldsymbol{\Psi}^T &= \boldsymbol{\Psi}\boldsymbol{\Psi}^T\mathbf{M}\mathbf{M}^\dagger\\ &= (\mathbf{I}_n - \mathbf{P})\mathbf{M}^\dagger\\ &= \mathbf{M}^\dagger - \boldsymbol{\Phi}_\perp\mathbf{G}^{-1}\boldsymbol{\Phi}_\perp^T - \mathbf{P}_0\mathbf{M}^\dagger, \qquad (4.61)\end{aligned}$$

which inserted into (4.57) yields

$$\mathbf{N}_J^{-1}\mathbf{r} = \boldsymbol{\Phi}\mathbf{A}_{11}^{-1}\boldsymbol{\Phi}^T\mathbf{r} + \frac{1}{\alpha}((\mathbf{I}_n - \mathbf{P}_0)\mathbf{M}^\dagger - \boldsymbol{\Phi}_\perp\mathbf{G}^{-1}\boldsymbol{\Phi}_\perp^T)\mathbf{r}.$$

We see that the computational costs are almost equal to the Jacobi preconditioner targeted a problem positive definite $\mathbf{M}$. Observe that $(\mathbf{I}_n - \mathbf{P}_0)\mathbf{M}^\dagger = \mathbf{M}^\dagger$ because the pseudo-inverse never has components in the null-space of $\mathbf{M}$. However, we prefer to keep the extended version because of implementation issues briefly explained in Appendix A.2.

Gauss-Seidel Preconditioning Revisited

We use equation (4.61) to avoid the explicit use of $\boldsymbol{\Psi}$ in (4.34)

$$\begin{aligned}\boldsymbol{\Psi}\mathbf{u}_2 &= \frac{1}{\alpha}\boldsymbol{\Psi}\boldsymbol{\Psi}(\mathbf{r} - \mathbf{K}^T\mathbf{K}\boldsymbol{\Phi}\mathbf{A}_{11}\boldsymbol{\Phi}^T\mathbf{r})\\ &= \frac{1}{\alpha}(\mathbf{M}^\dagger - \boldsymbol{\Phi}_\perp\mathbf{G}^{-1}\boldsymbol{\Phi}_\perp^T - \mathbf{P}_0\mathbf{M}^\dagger)(\mathbf{r} - \mathbf{K}^T\mathbf{K}\boldsymbol{\Phi}\mathbf{A}_{11}\boldsymbol{\Phi}^T\mathbf{r}),\end{aligned} \qquad (4.62)$$

while (4.35) remains unchanged

$$\boldsymbol{\Phi}\mathbf{u}_1 = \boldsymbol{\Phi}\mathbf{A}_{11}^{-1}\boldsymbol{\Phi}^T(\mathbf{r} - \mathbf{K}^T\mathbf{K}\boldsymbol{\Psi}\mathbf{u}_2). \qquad (4.63)$$

An algorithmic formulation is listed in Algorithm 7 and an actual Matlab implementation is located in Appendix A.4.2. We see that one preconditioning step involves two solutions with $\mathbf{A}_{11}$, one solution with $\mathbf{G}$ and one

"solution" with $\mathbf{M}$. The pseudo inverse is calculated by means of an QR factorization. In addition the preconditioning step involves two applications of $\mathbf{K}^T\mathbf{K}$, 6 applications of $\boldsymbol{\Phi}$ (A combination of $\mathbf{P}_0$, $\boldsymbol{\Phi}_\perp$ and $\boldsymbol{\Phi}_\perp^T$ is counted as two applications) and the application of $\mathbf{A}$ from the actual CG step.

Algorithm 7 Gauss-Seidel two-grid preconditioning, $\mathbf{u} = \mathbf{N}_{GS}^{-1}\mathbf{r}$ for system with semidefinite $\mathbf{L}^T\mathbf{L}$. Comments marks differences to Algorithm 5.

$\mathbf{v} = \boldsymbol{\Phi}\mathbf{A}_{11}^{-1}\boldsymbol{\Phi}^T\mathbf{r}$
$\mathbf{e} = \mathbf{r} - \mathbf{K}^T\mathbf{K}\mathbf{v}$
$\mathbf{w} = \mathbf{M}^\dagger\mathbf{e}$ % Pseudo inverse
$\mathbf{u} = \mathbf{w} - \mathbf{P}_0\mathbf{w} - \boldsymbol{\Phi}_\perp\mathbf{G}^{-1}\boldsymbol{\Phi}_\perp^T\mathbf{e}$ % Eq. (4.62)
$\mathbf{x}_W = \frac{1}{\alpha}\mathbf{x}$
$\mathbf{y} = \mathbf{r} - \mathbf{K}^T\mathbf{K}\mathbf{x}_W$
$\mathbf{x}_V = \boldsymbol{\Phi}\mathbf{A}_{11}^{-1}\boldsymbol{\Phi}^T\mathbf{y}$
$\mathbf{x} = \mathbf{x}_V + \mathbf{x}_W$

Schur Complement Method Revisited

It should come as no surprise that the modification of the Schur complement CG method follows the same track as the Jacobi and Gauss-Seidel preconditioners. Basicly we use the substitution of $\boldsymbol{\Psi}\boldsymbol{\Psi}^T$ from (4.61) instead of the similar transformation (4.27) used in (4.44) and (4.48). The first change is found for $\widetilde{\rho}$

$$\begin{aligned}\widetilde{\rho} &= \widetilde{\mathbf{r}}^T\widetilde{\mathbf{r}}\\ &= \mathbf{r}^T\boldsymbol{\Psi}\boldsymbol{\Psi}^T\mathbf{r}\\ &= \mathbf{r}^T\boldsymbol{\Psi}\boldsymbol{\Psi}^T\mathbf{M}\mathbf{M}^\dagger\mathbf{r}\\ &= \mathbf{r}^T(\mathbf{I}_n - \mathbf{P})\mathbf{M}^\dagger\mathbf{r}\\ &= \mathbf{r}^T(\mathbf{M}^\dagger - \boldsymbol{\Phi}_\perp\mathbf{G}^{-1}\boldsymbol{\Phi}_\perp^T\mathbf{M}\mathbf{M}^\dagger - \boldsymbol{\Phi}_0\boldsymbol{\Phi}_0^T\mathbf{M}^\dagger)\mathbf{r}\end{aligned}$$

and because of (4.59) we obtain

$$\widetilde{\rho} = \mathbf{r}^T(\mathbf{M}^\dagger - \boldsymbol{\Phi}_\perp\mathbf{G}^{-1}\boldsymbol{\Phi}_\perp^T - \boldsymbol{\Phi}_0\boldsymbol{\Phi}_0^T\mathbf{M}^\dagger)\mathbf{r}. \qquad (4.64)$$

Again we define the decent direction $\widetilde{\mathbf{d}}$ by

$$\widetilde{\mathbf{d}} = \boldsymbol{\Psi}^T\mathbf{d},$$

and applying the Schur complement yields

$$\begin{aligned}\mathbf{S}\widetilde{\mathbf{d}} &= \mathbf{\Psi}\mathbf{A}(\mathbf{I}_n - \mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{A})\mathbf{\Psi}\mathbf{\Psi}^T\mathbf{d} \\ &= \mathbf{\Psi}\mathbf{A}(\mathbf{I}_n - \mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{A})\mathbf{\Psi}\mathbf{\Psi}^T\mathbf{M}\mathbf{M}^\dagger\mathbf{d} \\ &= \mathbf{\Psi}\mathbf{A}(\mathbf{I}_n - \mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{A})(\mathbf{I}_n - \mathbf{\Phi}_\perp\mathbf{G}^{-1}\mathbf{\Phi}_\perp^T\mathbf{M} - \mathbf{\Phi}_0\mathbf{\Phi}_0^T)\mathbf{M}^\dagger\mathbf{d}\end{aligned} \tag{4.65}$$

We now focus on the middle part of (4.65)

$$\begin{gathered}(\mathbf{I}_n - \mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{A})(\mathbf{I}_n - \mathbf{\Phi}_\perp\mathbf{G}^{-1}\mathbf{\Phi}_\perp^T\mathbf{M} - \mathbf{\Phi}_0\mathbf{\Phi}_0^T) \\ = \\ \left(\mathbf{I}_n - \mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{A} - (\mathbf{I}_n - \mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{A})(\mathbf{\Phi}_\perp\mathbf{G}^{-1}\mathbf{\Phi}_\perp^T\mathbf{M} + \mathbf{\Phi}_0\mathbf{\Phi}_0^T)\right) \\ = \\ \mathbf{I}_n - \mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{A} - (\mathbf{I}_n - \mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{A})\left[\mathbf{\Phi}_0\ \mathbf{\Phi}_\perp\right]\begin{bmatrix}\mathbf{\Phi}_0^T \\ \mathbf{G}^{-1}\mathbf{\Phi}_\perp^T\mathbf{M}\end{bmatrix} \\ = \\ \mathbf{I}_n - \mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{A} - (\mathbf{I}_n - \mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{A})\mathbf{\Phi}\begin{bmatrix}\mathbf{\Phi}_0^T \\ \mathbf{G}^{-1}\mathbf{\Phi}_\perp^T\mathbf{M}\end{bmatrix} \\ = \\ \mathbf{I}_n - \mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{A} - (\mathbf{\Phi} - \mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{A}\mathbf{\Phi})\begin{bmatrix}\mathbf{\Phi}_0^T \\ \mathbf{G}^{-1}\mathbf{\Phi}_\perp^T\mathbf{M}\end{bmatrix} \\ = \\ \mathbf{I}_n - \mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{A} - (\mathbf{\Phi} - \mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{A}_{11})\begin{bmatrix}\mathbf{\Phi}_0^T \\ \mathbf{G}^{-1}\mathbf{\Phi}_\perp^T\mathbf{M}\end{bmatrix} \\ = \\ \mathbf{I}_n - \mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{A},\end{gathered} \tag{4.66}$$

which substituted into (4.65) yields

$$\mathbf{S}\widetilde{\mathbf{d}} = \mathbf{\Psi}^T\mathbf{A}(\mathbf{I}_n - \mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{A})\mathbf{M}^\dagger\mathbf{d}.$$

We now define $\mathbf{w} \in \mathbb{R}^n$ by

$$\mathbf{w} = \mathbf{A}(\mathbf{I}_n - \mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{A})\mathbf{M}^\dagger\mathbf{d},$$

and we have the simplified expression

$$\mathbf{S}\widetilde{\mathbf{d}} = \mathbf{\Psi}^T\mathbf{w} \tag{4.67}$$

Finally we calculate the denominator of $\widetilde{\beta} = \widetilde{\rho}/(\widetilde{\mathbf{d}}^T\mathbf{S}\widetilde{\mathbf{d}})$ from (4.67)

$$\begin{aligned}\widetilde{\mathbf{d}}^T\mathbf{S}\widetilde{\mathbf{d}} &= \mathbf{d}^T\mathbf{\Psi}\mathbf{\Psi}^T\mathbf{w} \\ &= \mathbf{d}^T\mathbf{\Psi}\mathbf{\Psi}^T\mathbf{M}\mathbf{M}^\dagger\mathbf{w} \\ &= \mathbf{d}^T(\mathbf{I}_n - \mathbf{\Phi}_\perp\mathbf{G}^{-1}\mathbf{\Phi}_\perp^T\mathbf{M} - \mathbf{V}_0\mathbf{V}_0^T)\mathbf{M}^\dagger\mathbf{w} \\ &= \mathbf{d}^T\mathbf{M}^\dagger\mathbf{w} - \mathbf{d}^T(\mathbf{\Phi}_\perp\mathbf{G}^{-1}\mathbf{\Phi}_\perp^T\mathbf{M} - \mathbf{V}_0\mathbf{V}_0^T)\mathbf{M}^\dagger\mathbf{w}\end{aligned}$$

The second term is rewritten as follows

$$\begin{gathered}\mathbf{d}^T(\mathbf{\Phi}_\perp\mathbf{G}^{-1}\mathbf{\Phi}_\perp^T\mathbf{M} - \mathbf{V}_0\mathbf{V}_0^T)\mathbf{M}^\dagger\mathbf{w} \\ = \\ \mathbf{d}^T(\mathbf{\Phi}_\perp\mathbf{G}^{-1}\mathbf{\Phi}_\perp^T)\mathbf{w} \\ = \\ \mathbf{d}^T(\mathbf{\Phi}_\perp\mathbf{G}^{-1}\mathbf{\Phi}_\perp^T)\mathbf{A}(\mathbf{I}_n - \mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{A})\mathbf{M}^\dagger\mathbf{d} \\ = \\ \mathbf{d}^T\left[\mathbf{0}_{n,k}\ \mathbf{\Phi}_\perp\mathbf{G}^{-1}\right]\begin{bmatrix}\mathbf{\Phi}_0^T \\ \mathbf{\Phi}_\perp^T\end{bmatrix}\mathbf{A}(\mathbf{I}_n - \mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{A})\mathbf{M}^\dagger\mathbf{d} \\ = \\ \mathbf{d}^T\left[\mathbf{0}_{n,k}\ \mathbf{\Phi}_\perp\mathbf{G}^{-1}\right]\mathbf{\Phi}^T\mathbf{A}(\mathbf{I}_n - \mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{A})\mathbf{M}^\dagger\mathbf{d} \\ = \\ \mathbf{d}^T\left[\mathbf{0}_{n,k}\ \mathbf{\Phi}_\perp\mathbf{G}^{-1}\right](\mathbf{\Phi}^T\mathbf{A} - \mathbf{\Phi}^T\mathbf{A}\mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{A})\mathbf{M}^\dagger\mathbf{d} \\ = \\ \mathbf{d}^T\left[\mathbf{0}_{n,k}\ \mathbf{\Phi}_\perp\mathbf{G}^{-1}\right](\mathbf{\Phi}^T\mathbf{A} - \mathbf{A}_{11}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{A})\mathbf{M}^\dagger\mathbf{d} \\ = \\ \mathbf{d}^T\left[\mathbf{0}_{n,k}\ \mathbf{\Phi}_\perp\mathbf{G}^{-1}\right](\mathbf{0}_{k,n})\mathbf{M}^\dagger\mathbf{d} \\ = \\ 0\end{gathered} \tag{4.68}$$

and we obtain

$$\widetilde{\mathbf{d}}^T\mathbf{S}\widetilde{\mathbf{d}} = \mathbf{d}^T\mathbf{M}^\dagger\mathbf{w}. \tag{4.69}$$

The result compared to the SPD-version is a change from the inverse of $\mathbf{M}$ to the pseudo inverse $\mathbf{M}^\dagger$ and one extra projection operation. The algorithm is listed in Algorithm 8 and the Matlab program can be found in Appendix A.4.3.

Algorithm 8 Schur complement CG for semidefinite $\mathbf{M} = \mathbf{L}^T\mathbf{L}$. Changes w.r.t Algorithm 6 is marked by comments.

$\mathbf{u}_{(0)} = \mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{b}$
$\mathbf{r}_{(0)} = \mathbf{b} - \mathbf{A}\mathbf{u}_{(0)}$
$\mathbf{q}_{(0)} = \mathbf{M}^\dagger\mathbf{r}_{(0)}$ % Pseudo inverse
$\mathbf{y}_{(0)} = \mathbf{q}_{(0)} - \mathbf{P}_\perp\mathbf{G}^{-1}\mathbf{P}_\perp^T\mathbf{r}_{(0)} - \mathbf{P}_0\mathbf{q}_{(0)}$ % Eq. (4.61)
$\mathbf{d}_{(0)} = \mathbf{r}_{(0)}$
$\mathbf{z}_{(0)} = \mathbf{y}_{(0)}$
$\rho_{(0)} = \mathbf{y}_{(0)}^T\mathbf{r}_{(0)}$
$i \leftarrow 0$
repeat
  $\mathbf{v}_{(i)} = (\mathbf{I}_n - \mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{A})\mathbf{z}_{(i)}$
  $\mathbf{w}_{(i)} = \mathbf{A}\mathbf{v}_{(i)}$
  $\mathbf{f}_{(i)} = \mathbf{M}^\dagger\mathbf{w}_{(i)}$ % Pseudo inverse
  $\mathbf{g}_{(i)} = \mathbf{f}_{(i)} - \mathbf{P}_\perp\mathbf{G}^{-1}\mathbf{P}_\perp^T\mathbf{w}_{(i)} - \mathbf{P}_0\mathbf{f}_{(i)}$ % Subtraction of null space
  $\beta_{(i)} = \rho_{(i)}/(\mathbf{d}_{(i)}^T\mathbf{g}_{(i)})$
  $\mathbf{u}_{(i+1)} = \mathbf{u}_{(i)} + \beta_{(i)}\mathbf{v}_{(i)}$
  $\mathbf{r}_{(i+1)} = \mathbf{r}_{(i)} - \beta_{(i)}\mathbf{w}_{(i)}$
  $\mathbf{y}_{(i+1)} = \mathbf{y}_{(i)} - \beta_{(i)}\mathbf{g}_{(i)}$
  $\rho_{(i+1)} = \mathbf{y}_{(i+1)}^T\mathbf{r}_{(i+1)}$
  $\gamma_{(i)} = \rho_{(i+1)}/\rho_{(i)}$
  $\mathbf{d}_{(i+1)} = \mathbf{r}_{(i+1)} + \gamma_{(i)}\mathbf{d}_{(i)}$
  $\mathbf{z}_{(i+1)} = \mathbf{y}_{(i+1)} + \gamma_{(i)}\mathbf{z}_{(i)}$
  $i \leftarrow i + 1$
until stop

A count of operations tells that Algorithm 8 needs one solution with $\mathbf{A}_{11}$, one solution with $\mathbf{G}$ and one solution with $\mathbf{M}$. Furthermore we count two applications of $\mathbf{A}$, one and 6 applications of $\mathbf{\Phi}$ or $\mathbf{\Phi}^T$ ($\mathbf{P}_0$ and 2 times $\mathbf{P}_\perp$ counting as 4 applications).

4.4 A Theoretical Comparison

This section describes the expectations we have regarding the algorithms both in terms of computational cost and convergence speed from a theoretical point of view. The theoretical results presented here are in Chapter 6 compared to actual implementations applied to a series of test problems.

4.4.1 Computational Costs

Table 4.1 contains a summary of the computational costs of one iteration for the two-grid methods. Jacobi is very simple and it comes as no surprise that it is cheaper than both Gauss-Seidel and Schur complement. The semidefinite $\mathbf{M}$ case involves two extra applications of $\mathbf{\Phi}$ for both Schur CG and Jacobi while Schur CG also have one solution with $\mathbf{G}$ not found in the SPD case. The GS preconditioner is equally expensive in both cases.

	Positive definite $\mathbf{M}$			Semi definite $\mathbf{M}$			
Matrix	Jacobi	GS	Schur	Jacobi	GS	Schur	CGLS
$\mathbf{K}^T\mathbf{K}$	1	3	2	2	3	2	1
$\mathbf{L}^T\mathbf{L}$	1	1	2	1	1	2	1
$\mathbf{\Phi}$	2	6	2	4	6	4	0
$\mathbf{A}_{11}^{-1}$	1	2	1	1	2	1	0
$\mathbf{G}^{-1}$	1	1	0	1	1	1	0
$\mathbf{M}^{-1}$	1	1	1	0	0	0	0
$\mathbf{M}^\dagger$	0	0	0	1	1	1	0

Table 4.1: *Summary of of matrix-vector products and solutions in each algorithm. An inverse matrix implies a solution with the matrix as coefficient matrix. CGLS is assumed to work on the regularized system. Initial work is not included.*

Assuming that the cost of applying $\mathbf{K}^T\mathbf{K}$ is the dominant factor the operation count tells us that the improvement per iteration should be at least twice as good as the standard CGLS algorithm in the case of Jacobi and Schur CG, while GS must triple the improvement per iteration. Whether these requirements are met or not is the subject of Section 6.8. The cost of $\mathbf{\Phi}$ is the the subspace dimension k and would in a matrix implementation be $2kn$ flops (floating point instructions). The solutions with $\mathbf{A}_{11}$ and $\mathbf{G}$ both imply two backsubstitutions of each k^2 — one of the reasons to keep the subspace dimension low.

Before the iterations can start it is necessary to perform a number of tasks:

- The creation of $\mathbf{A}_{11}$ uses $3km(n+k)$ flops with dense matrix calculations.
- The factorization of $\mathbf{A}_{11}$ uses $k^3/3$ flops if we choose a Cholesky factorization.
- The cost of creating $\mathbf{G} = \mathbf{\Phi}^T\mathbf{L}^T\mathbf{L}\mathbf{\Phi}$ depends on the dimension and the sparsity of $\mathbf{L}$ — $\mathbf{L}$ is in most applications very sparse and the evaluation of $\mathbf{G}$ can be neglected.
- The factorization of $\mathbf{G}$ uses $k^3/3$ flops using a Cholesky factorization.
- The factorization of $\mathbf{M} = \mathbf{L}^T\mathbf{L}$ also depends on the sparsity of $\mathbf{L}$ and can be neglected when using a QR factorization. The pseudo-inverse is also obtainable from a QR factorization of $\mathbf{L}$.

If $\mathbf{L} = \mathbf{I}_n$ and the columns of $\mathbf{\Phi}$ are orthogonal only the creation and factorization of $\mathbf{A}_{11}$ remain because $\mathbf{G} = \mathbf{I}_k$ and $\mathbf{M} = \mathbf{I}_n$.

4.4.2 Convergence Properties

Past experience (e.g. [2]) shows that a Gauss-Seidel preconditioner performs better than a Jacobi preconditioner for most equation systems. In the following we show that the Schur complement and Gauss-Seidel two-grid preconditioned system both have better bounds on their condition number compared to that of the Jacobi two-grid preconditioned.

Before we proceed we introduce two quantities, $\gamma_{\mathbf{\Phi}}$ and $\gamma_{\mathbf{\Psi}}$, that are important to the convergence results

$$\gamma_{\mathbf{\Phi}} = \|\mathbf{KP}\|_{\mathbf{M}} = \|\mathbf{K}|_{\mathcal{V}_k}\|_{\mathbf{M}} = \sup_{\mathbf{u}\in\mathcal{V}_k\setminus\mathcal{N}(\mathbf{M})} \frac{\|\mathbf{Ku}\|_2}{\|\mathbf{u}\|_{\mathbf{M}}}, \tag{4.70}$$

$$\gamma_{\mathbf{\Psi}} = \|\mathbf{KQ}\|_{\mathbf{M}} = \|\mathbf{K}|_{\mathcal{W}_k}\|_{\mathbf{M}} = \sup_{\mathbf{u}\in\mathcal{W}_k\setminus\{\mathbf{0}\}} \frac{\|\mathbf{Ku}\|_2}{\|\mathbf{u}\|_{\mathbf{M}}} \tag{4.71}$$

Intuitively (but not correctly) $\gamma_{\mathbf{\Psi}}$ equals the largest singular value (the general singular value if $\mathbf{L} \neq \mathbf{I}_n$) that has its vector in the subspace $\mathcal{W}_k$. Hence a subspace $\mathcal{V}_k$ including the right singular vectors belonging to the large singular values implies that $\gamma_{\mathbf{\Psi}}$ is small because only singular vectors with small singular values reside in $\mathcal{W}_k$. A geometric interpretation in three dimensions is illustrated in Figure 4.2.

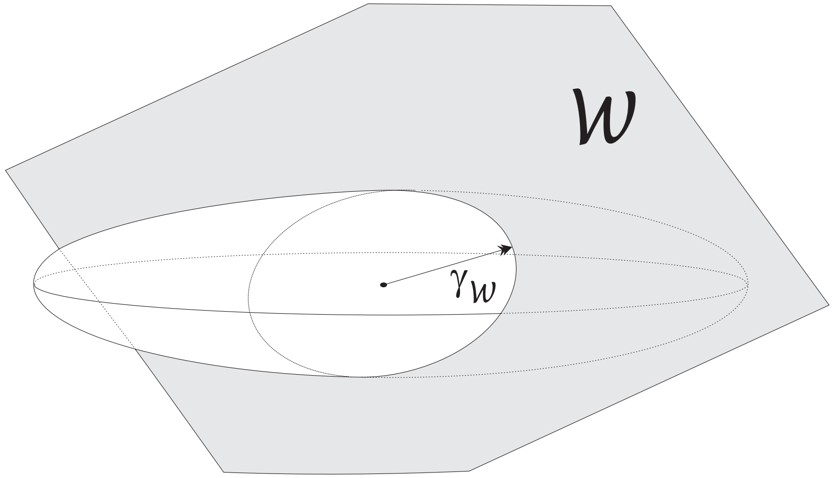


Figure 4.2: 3-D operator norm with restriction. The 3D unit ball (in $\mathbf{M}$-norm to be precise) is mapped onto an ellipsoid with semi-axes according to the singular vales of the operator. The 2D subspace $\mathcal{W}_k$ intersects with the ellipsoid in a ellipsis. The largest semi-axis of this ellipsis equals the restricted operator norm $\|\mathbf{K}|_{\mathcal{W}_k}\| = \gamma_{\mathbf{\Psi}}$, while the distance from the origo orthogonal to the $\mathcal{W}_k$ equals $\gamma_{\mathbf{\Phi}}$.

Jacobi Preconditioning

To find the condition number of the Jacobi preconditioned system we initially construct the preconditioned matrix from (4.19) and (4.23) using the square root of $\mathbf{N}_J$

$$\begin{aligned}
\mathbf{J} &= \mathbf{N}_J^{-1/2}\,[\mathbf{\Phi}\ \mathbf{\Psi}]^T\,\mathbf{A}\,[\mathbf{\Phi}\ \mathbf{\Psi}]\,\mathbf{N}_J^{-1/2} \\
&= \begin{bmatrix} \mathbf{A}_{11}^{-1/2} & \mathbf{0} \\ \mathbf{0} & 1/\sqrt{\alpha}\mathbf{I}_{n-k} \end{bmatrix} \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix} \begin{bmatrix} \mathbf{A}_{11}^{-1/2} & \mathbf{0} \\ \mathbf{0} & 1/\sqrt{\alpha}\mathbf{I}_{n-k} \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{I}_k & \mathbf{A}_{11}^{-1/2}\mathbf{A}_1 2/\sqrt{\alpha} \\ \mathbf{A}_{21}\mathbf{A}_{11}^{-1/2}/\sqrt{\alpha} & \mathbf{A}_{22}/\alpha \end{bmatrix}. \qquad (4.72)
\end{aligned}$$

Now assume we know an eigenpair $(\lambda, \mathbf{u})$ to (4.72) and let $\mathbf{u} = [\mathbf{u}_1^T\ \mathbf{u}_2^T]^T$ denote the block components corresponding to the given subspaces scaled such that $\|\mathbf{u}_2\|_2 = 1$, implying that $\|[\mathbf{\Phi}\ \mathbf{\Psi}][\mathbf{0}_{k,1}\ \mathbf{u}_2]\|_{\mathbf{M}} = 1$. For convenience we define $\mathbf{v} = \mathbf{\Phi}\mathbf{u}_1$ and $\mathbf{w} = \mathbf{\Psi}\mathbf{u}_2$. Multiplying the system (4.72) with its eigenvalue yields two equations

$$\begin{aligned}
\mathbf{J}\mathbf{u} &= \begin{bmatrix} \mathbf{I}_k & \mathbf{A}_{11}^{-1/2}\mathbf{A}_1 2/\sqrt{\alpha} \\ \mathbf{A}_{21}\mathbf{A}_{11}^{-1/2}/\sqrt{\alpha} & \mathbf{A}_{22}/\alpha \end{bmatrix} \begin{bmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \end{bmatrix} \\
&= \lambda\mathbf{u} \\
&\Updownarrow \\
\lambda\mathbf{u}_1 &= \mathbf{u}_1 + \mathbf{A}_{11}^{-1/2}\mathbf{A}_{12}\mathbf{u}_2/\sqrt{\alpha}, \qquad (4.73) \\
\lambda\mathbf{u}_2 &= \mathbf{A}_{12}\mathbf{A}_{11}^{-1/2}\mathbf{u}_1/\sqrt{\alpha} + \mathbf{A}_{22}\mathbf{u}_2/\alpha. \qquad (4.74)
\end{aligned}$$

Isolating $\mathbf{u}_1$ in (4.73) and inserting the result into (4.74) gives

$$\begin{aligned}
\lambda\mathbf{u}_2 &= \frac{\mathbf{A}_{21}\mathbf{A}_{11}^{-1/2}\mathbf{A}_{11}^{-1/2}\mathbf{A}_{12}}{\alpha(\lambda-1)} + \mathbf{A}_{22}\mathbf{u}_2/\alpha \\
&\Downarrow \\
\lambda(\lambda-1)\mathbf{u}_2 &= \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{A}_{12}/\alpha + (\lambda-1)\mathbf{A}_{22}\mathbf{u}_2/\alpha
\end{aligned}$$

Finally we premultiply with $\mathbf{u}_2^T$

$$\begin{aligned}
\lambda(\lambda-1)\mathbf{u}_2^T\mathbf{u}_2 &= \mathbf{u}_2^T\mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{A}_{12}/\alpha + (\lambda-1)\mathbf{u}_2^T\mathbf{A}_{22}\mathbf{u}_2/\alpha \\
&\Updownarrow \\
\lambda^2 - \lambda &= q + (\lambda-1)\mathbf{u}_2^T\mathbf{A}_{22}\mathbf{u}_2/\alpha + (\lambda-1) - (\lambda-1) \\
&\Updownarrow \\
\lambda^2 - \lambda &= q + (\lambda-1)(\mathbf{u}_2^T\mathbf{A}_{22}\mathbf{u}_2/\alpha - 1) + (\lambda-1) \\
&\Updownarrow \\
\lambda^2 - 2\lambda + 1 - p(\lambda-1) - q &= 0,
\end{aligned} \qquad (4.75)$$

where p and q are short for

$$p = \frac{\mathbf{u}_2^T\mathbf{A}_{22}\mathbf{u}_2}{\alpha} - 1 \quad \text{and} \quad q = \frac{\mathbf{u}_2^T\mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{A}_{12}\mathbf{u}_2}{\alpha}. \qquad (4.76)$$

Observe that (4.75) is a second degree polynomial where λ is a root if $(\lambda, \mathbf{u})$ is an eigenpair. We will now find bounds for the possible roots of the polynomial and as a consequence we also find an upper bound for the condition number of $\mathbf{J}$.

First we find bounds for p

$$\begin{aligned}
p &= \frac{\mathbf{u}_2^T\mathbf{A}_{22}\mathbf{u}_2}{\alpha} - 1 \\
&= \frac{\mathbf{u}_2^T\mathbf{A}_{22}\mathbf{u}_2 - \alpha\mathbf{u}_2^T\mathbf{u}_2}{\alpha} \\
&= \frac{\mathbf{u}_2^T(\mathbf{A}_{22} - \alpha\mathbf{I}_{n-k})\mathbf{u}_2}{\alpha} \\
&= \frac{\mathbf{w}^T\mathbf{K}^T\mathbf{K}\mathbf{w}}{\alpha} \\
&= \frac{\|\mathbf{K}\mathbf{w}\|_2^2}{\alpha} \\
&\leq \gamma_{\mathbf{\Psi}}^2/\alpha, \qquad (4.77)
\end{aligned}$$

and because p is non-negative we have the following bound

$$0 \leq p \leq \gamma_{\mathbf{\Psi}}^2/\alpha.$$

Now we turn to q. From (4.76) and the definition of the block decomposition we get

$$\begin{aligned} q &= \frac{\mathbf{u}_2^T \mathbf{\Psi}^T \mathbf{K}^T \mathbf{K}\mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{K}^T\mathbf{K}\mathbf{\Psi}\mathbf{u}_2}{\alpha} \\ &= \frac{(\mathbf{K}\mathbf{\Psi}\mathbf{u}_2)^T(\mathbf{K}\mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{K}^T)(\mathbf{K}\mathbf{\Psi}\mathbf{u}_2)}{\alpha} \\ &= \frac{(\mathbf{K}\mathbf{\Psi}\mathbf{u}_2)^T\mathbf{H}(\mathbf{K}\mathbf{\Psi}\mathbf{u}_2)}{\alpha} \end{aligned} \tag{4.78}$$

where $\mathbf{H}$ is defined as

$$\begin{aligned} \mathbf{H} &= \mathbf{K}\mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{K}^T \\ &= \mathbf{K}\mathbf{\Phi}(\mathbf{\Phi}^T\mathbf{K}^T\mathbf{K}\mathbf{\Phi} + \alpha\mathbf{\Phi}^T\mathbf{M}\mathbf{\Phi})^{-1}\mathbf{\Phi}^T\mathbf{K}^T \end{aligned} \tag{4.79}$$

Substituting[2] $\mathbf{G} = \mathbf{\Phi}^T\mathbf{M}\mathbf{\Phi}$ and $\mathbf{T} = \mathbf{K}\mathbf{\Phi}\mathbf{G}^{-1/2} \in \mathbb{R}^{n\times k}$ and using the "insert the identity" trick simplifies equation (4.79)

$$\begin{aligned} \mathbf{H} &= \mathbf{K}\mathbf{\Phi}\mathbf{G}^{-1/2}\mathbf{G}^{1/2}(\mathbf{\Phi}^T\mathbf{K}^T\mathbf{K}\mathbf{\Phi} + \alpha\mathbf{G})^{-1}\mathbf{G}^{1/2}\mathbf{G}^{-1/2}\mathbf{\Phi}^T\mathbf{K}^T \\ &= \mathbf{K}\mathbf{\Phi}\mathbf{G}^{-1/2}(\mathbf{G}^{-1/2}\mathbf{\Phi}^T\mathbf{K}^T\mathbf{K}\mathbf{\Phi}\mathbf{G}^{-1/2} + \alpha\mathbf{I}_k)^{-1}\mathbf{G}^{-1/2}\mathbf{\Phi}^T\mathbf{K}^T \\ &= \mathbf{T}(\mathbf{T}^T\mathbf{T} + \alpha\mathbf{I}_k)^{-1}\mathbf{T}^T \end{aligned}$$

It is obvious that $(\mathbf{T}^T\mathbf{T} + \alpha\mathbf{I})$ is SPD and we conclude that $\mathbf{H}$ is SPD. Let $\mathbf{T} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ be the SVD decomposition of $\mathbf{T}$ and consider the bound

$$\begin{aligned} \|\mathbf{H}\|_2 &= \|\mathbf{T}(\mathbf{T}^T\mathbf{T} + \alpha\mathbf{I}_k)^{-1}\mathbf{T}^T\|_2 \\ &= \|\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T(\mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^T + \alpha\mathbf{I}_k)^{-1}\mathbf{V}\mathbf{\Sigma}\mathbf{U}^T\|_2 \\ &= \|\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T\mathbf{V}(\mathbf{\Sigma}^2 + \alpha\mathbf{I}_k)^{-1}\mathbf{V}^T\mathbf{V}\mathbf{\Sigma}\mathbf{U}^T\|_2 \\ &= \|\mathbf{\Sigma}(\mathbf{\Sigma}^2 + \alpha\mathbf{I}_k)^{-1}\mathbf{\Sigma}\|_2 \\ &= \|\mathbf{\Sigma}(\mathrm{diag}(\sigma_i^2 + \alpha))^{-1}\mathbf{\Sigma}\|_2 \\ &= \left\|\mathrm{diag}\left(\frac{\sigma_i^2}{\sigma_i^2+\alpha}\right)\right\|_2 \\ &\leq 1. \end{aligned} \tag{4.80}$$

[2]This substitution assumes $\mathbf{M}$ to be SPD and invertible — otherwise we are forced to use the same tricks to move $\mathbf{\Phi}$ around as in (4.66) and (4.68) which would double the page-count of this section.

Because $\|\mathbf{H}\|_2 \leq 1$ it follows that $(\mathbf{K}\mathbf{\Psi}\mathbf{u}_2)^T\mathbf{H}(\mathbf{K}\mathbf{\Psi}\mathbf{u}_2) \leq (\mathbf{K}\mathbf{\Psi}\mathbf{u}_2)^T(\mathbf{K}\mathbf{\Psi}\mathbf{u}_2)$. Thus we obtain from (4.78) and (4.80) the bound

$$0 \leq q \leq \|\mathbf{H}\|_2\|\mathbf{K}\mathbf{u}_2\|_2/\alpha \leq \gamma_{\mathbf{\Psi}}^2/\alpha$$

Finally we need to find a bound for $p-q$. First use (4.77) and (4.78) to form

$$p - q = (\mathbf{K}\mathbf{\Psi}\mathbf{u}_2)^T(\mathbf{I}_n - \mathbf{H})\mathbf{K}\mathbf{\Psi}\mathbf{u}_2/\alpha. \tag{4.81}$$

The subtraction $\mathbf{I}_n - \mathbf{H}$ is rewritten with help from B.1 (in the appendix)

$$\begin{aligned} \mathbf{I}_n - \mathbf{H} &= \mathbf{I}_n - \mathbf{T}(\mathbf{T}^T\mathbf{T} + \alpha\mathbf{I}_n)^{-1}\mathbf{T}^T \\ &= \mathbf{I}_n - \mathbf{T}\mathbf{T}^T(\mathbf{T}\mathbf{T}^T + \alpha\mathbf{I}_n)^{-1} \\ &= \mathbf{I}_n + \alpha\mathbf{I}_n(\mathbf{T}\mathbf{T}^T + \alpha\mathbf{I}_n)^{-1} \\ &\qquad - \alpha\mathbf{I}_n(\mathbf{T}\mathbf{T}^T + \alpha\mathbf{I}_n)^{-1} - \mathbf{T}\mathbf{T}^T(\mathbf{T}\mathbf{T}^T + \alpha\mathbf{I}_n)^{-1} \\ &= \mathbf{I}_n + \alpha\mathbf{I}_n(\mathbf{T}\mathbf{T}^T + \alpha\mathbf{I}_n)^{-1} - (\alpha\mathbf{I}_n + \mathbf{T}\mathbf{T}^T)(\mathbf{T}\mathbf{T}^T + \alpha\mathbf{I}_n)^{-1} \\ &= \mathbf{I}_n + \alpha\mathbf{I}_n(\mathbf{T}\mathbf{T}^T + \alpha\mathbf{I}_n)^{-1} - \mathbf{I}_n \\ &= \alpha\mathbf{I}_n(\mathbf{T}\mathbf{T}^T + \alpha\mathbf{I}_n)^{-1}, \end{aligned}$$

which in a combination of (4.81) and an eigenvalue decomposition of $(\mathbf{T}\mathbf{T}^{\mathbf{T}} + \alpha\mathbf{I}) = \dot{\mathbf{V}}\dot{\mathbf{\Sigma}}^2\dot{\mathbf{V}}^T$ (written in terms of the SVD) and a temporary rewrite $\mathbf{y} = \dot{\mathbf{V}}^T\mathbf{K}\mathbf{\Psi}\mathbf{u}_2$ yield

$$\begin{aligned} p - q &= (\mathbf{K}\mathbf{\Psi}\mathbf{u}_2)^T(\mathbf{T}\mathbf{T}^T + \alpha\mathbf{I}_n)^{-1}\mathbf{K}\mathbf{\Psi}\mathbf{u}_2 \\ &= (\mathbf{K}\mathbf{\Psi}\mathbf{u}_2)^T(\dot{\mathbf{V}}\dot{\mathbf{\Sigma}}^2\dot{\mathbf{V}}^T)^{-1}\mathbf{K}\mathbf{\Psi}\mathbf{u}_2) \\ &= \|(\dot{\mathbf{V}}^T\mathbf{K}\mathbf{\Psi}\mathbf{u}_2)^T\dot{\mathbf{\Sigma}}^{-2}(\dot{\mathbf{V}}^T\mathbf{K}\mathbf{\Psi}\mathbf{u}_2)\|_2^2 \\ &= \sum_{i=1}^n \dot{\sigma}_i^{-2}y_i^2 \\ &\geq \sum_{i=1}^n \dot{\sigma}_1^{-2}y_i^2 \\ &= \frac{\|\dot{\mathbf{V}}^T\mathbf{K}\mathbf{\Psi}\mathbf{u}_2\|_2^2}{\|\mathbf{T}\mathbf{T}^T + \alpha\mathbf{I}_n\|_2} \\ &= \frac{\|\mathbf{K}\mathbf{\Psi}\mathbf{u}_2\|_2^2}{\|\mathbf{T}\mathbf{T}^T + \alpha\mathbf{I}_n\|_2}. \end{aligned} \tag{4.82}$$

Now we have the basic bounds on q and p and it is possible to find bounds on the roots of equation (4.75). We know p and q are nonnegative and an upper bound is now easily found

$$\begin{aligned}(\lambda - 1) &= \frac{-p \pm \sqrt{p^2 - 4q}}{2} \\ &\Downarrow \\ \lambda &\leq 1 + \frac{p + \sqrt{p^2 + 4q}}{2} \\ &\leq 1 + p + \sqrt{q} \\ &\leq \frac{\gamma_{\boldsymbol{\Psi}}^2}{\alpha} + \frac{\gamma_{\boldsymbol{\Psi}}}{\sqrt{\alpha}} + 1 \end{aligned} \tag{4.83}$$

Likewise we find an lower bound for λ

$$\begin{aligned}\lambda &= 1 + \frac{p \pm \sqrt{p^2 - 4q}}{2} \\ &\geq 1 + \frac{p + \sqrt{p^2 - 4q}}{2} \\ &= 1 + \frac{4q}{2(p + \sqrt{p^2 - 4q})} \\ &\geq 1 - \frac{q}{p} \\ &= \frac{p-q}{p}. \end{aligned} \tag{4.84}$$

and inserting the definitions of $p - q$ and p gives

$$\begin{aligned}\frac{p-q}{p} &= \frac{\|\mathbf{K}\boldsymbol{\Psi}\mathbf{u}_2\|_2^2 \alpha}{\|\mathbf{T}\mathbf{T}^T + \alpha \mathbf{I}_n\|_2 \|\mathbf{K}\boldsymbol{\Psi}\mathbf{u}_2\|_2^2} \\ &= \frac{\alpha}{\|\mathbf{T}\mathbf{T}^T + \alpha \mathbf{I}_n\|_2} \\ &= \frac{\alpha}{\|\mathbf{T}\|_2^2 + \alpha}\end{aligned}$$

The definition of $\mathbf{T} = K\boldsymbol{\Phi}\mathbf{G}^{-1/2}$ tells us ($\bar{\mathbf{z}} = \mathbf{G}^{-1/2}\mathbf{z}$)

$$\begin{aligned}\|\mathbf{T}\|_2^2 &= \sup_{\mathbf{z}\in\mathbb{R}^n\setminus\{\mathbf{0}\}} \frac{\|K\boldsymbol{\Phi}\mathbf{G}^{-1/2}\mathbf{z}\|_2^2}{\|\mathbf{z}\|_2^2} \\ &= \sup_{\bar{\mathbf{z}}\in\mathbb{R}^k\setminus\{\mathbf{0}\}} \frac{\|K\boldsymbol{\Phi}\bar{\mathbf{z}}\|_2^2}{\bar{\mathbf{z}}^T\mathbf{G}\bar{\mathbf{z}}} \\ &= \gamma_{\boldsymbol{\Phi}}.\end{aligned} \tag{4.85}$$

The lower bound is now derived to be

$$\lambda \geq \frac{\alpha}{\alpha + \gamma_{\boldsymbol{\Phi}}},$$

and combined with the upper bound we find a condition number bound of the Jacobi preconditioned system

$$\mathrm{cond}(\mathbf{J}) \leq \frac{(\gamma_{\boldsymbol{\Psi}}/\alpha + \gamma_{\boldsymbol{\Psi}}/\sqrt{\alpha} + 1)\,(\alpha + \gamma_{\boldsymbol{\Phi}})}{\alpha}.$$

If α is small we get

$$\mathrm{cond}(\mathbf{J}) = \mathcal{O}(\alpha^{-2}), \quad \alpha \ll 1, \tag{4.86}$$

while a large α yields the estimation

$$\mathrm{cond}(\mathbf{J}) = 1 + \mathcal{O}(\gamma_{\boldsymbol{\Psi}}/\sqrt{\alpha}), \quad \alpha \gg \gamma_{\boldsymbol{\Psi}}$$

Here we get the first indication that the Jacobi method are unsuitable for our purpose because the condition number in the usual case, a small α, is proportional to $1/\alpha^2$. The unpreconditioned system has a condition number proportional to $1/\alpha$ and should therefore converge faster.

4.4.3 Schur Complement Condition Number

The condition number of the Schur complement is of vital importance to Schur complement CG. Later we will see that the Schur complement also finds its way into the condition number of the Gauss-Seidel preconditioned system. We bound the condition number using bounds for the largest and smallest eigenvalue of $\mathbf{S}$.

According to Axelsson [1, Theorem 3.8] we have for any $\mathbf{y} \in \mathbb{R}^{n-k}$ (a vector in "Schur complement space") the following equality

$$\mathbf{y}^T\mathbf{S}\mathbf{y} = \inf_{\mathbf{u}\in\mathbb{R}^k} [\mathbf{u}^T\mathbf{y}^T] \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ \mathbf{y} \end{bmatrix},$$

which combined with the definition of the transformed system (4.12)

$$\begin{aligned} \mathbf{y}^T\mathbf{S}\mathbf{y} &= \inf_{\mathbf{u}\in\mathbb{R}^k} [\mathbf{u}^T\mathbf{y}^T] \begin{bmatrix} \mathbf{\Phi}^T \\ \mathbf{\Psi}^T \end{bmatrix} \mathbf{A}\,[\mathbf{\Phi}\ \mathbf{\Psi}] \begin{bmatrix} \mathbf{u} \\ \mathbf{y} \end{bmatrix} \\ &= \inf_{\mathbf{u}\in\mathbb{R}^k} \left(\mathbf{u}^T\mathbf{\Phi}^T + \mathbf{y}^T\mathbf{\Psi}^T\right)\left(\mathbf{K}^T\mathbf{K} + \alpha\mathbf{M}\right)\left(\mathbf{\Phi}\mathbf{u} + \mathbf{\Psi}\mathbf{y}\right). \quad (4.87) \end{aligned}$$

We now introduce $\mathbf{v} = \mathbf{\Phi}\mathbf{u}$ and $\mathbf{w} = \mathbf{\Psi}\mathbf{y}$ and (4.87) is simplified to

$$\begin{aligned} \mathbf{y}^T\mathbf{S}\mathbf{y} &= \inf_{\mathbf{v}\in\mathcal{V}_k} \left(\mathbf{v}^T + \mathbf{w}^T\right)\left(\mathbf{K}^T\mathbf{K} + \alpha\mathbf{M}\right)(\mathbf{v} + \mathbf{w}) \\ &= \inf_{\mathbf{v}\in\mathcal{V}_k} \Big(\mathbf{v}^T\mathbf{K}^T\mathbf{K}\mathbf{v} + \alpha\mathbf{v}^T\mathbf{M}\mathbf{v} + \\ &\qquad \mathbf{v}^T\mathbf{K}^T\mathbf{K}\mathbf{w} + \alpha\mathbf{v}^T\mathbf{M}\mathbf{w} + \\ &\qquad \mathbf{w}^T\mathbf{K}^T\mathbf{K}\mathbf{v} + \alpha\mathbf{w}^T\mathbf{M}\mathbf{v} + \\ &\qquad \mathbf{w}^T\mathbf{K}^T\mathbf{K}\mathbf{w} + \alpha\mathbf{w}^T\mathbf{M}\mathbf{w}\Big) \\ &= \inf_{\mathbf{v}\in\mathcal{V}_k} \Big(\mathbf{v}^T\mathbf{K}^T\mathbf{K}\mathbf{v} + \alpha\mathbf{v}^T\mathbf{M}\mathbf{v} + \\ &\qquad \mathbf{v}^T\mathbf{K}^T\mathbf{K}\mathbf{w} + 0 + \\ &\qquad \mathbf{w}^T\mathbf{K}^T\mathbf{K}\mathbf{v} + 0 + \\ &\qquad \mathbf{w}^T\mathbf{K}^T\mathbf{K}\mathbf{w} + \alpha\mathbf{w}^T\mathbf{M}\mathbf{w}\Big) \\ &= \inf_{\mathbf{v}\in\mathcal{V}_k} \Big((\mathbf{v}^T + \mathbf{w}^T)\mathbf{K}^T\mathbf{K}(\mathbf{v} + \mathbf{w}) + \alpha\mathbf{v}^T\mathbf{M}\mathbf{v} + \alpha\mathbf{w}^T\mathbf{M}\mathbf{w}\Big) \\ &= \inf_{\mathbf{v}\in\mathcal{V}_k} \Big(\|\mathbf{K}(\mathbf{v} + \mathbf{w})\|_2^2 + \alpha\|\mathbf{v}\|_{\mathbf{M}}^2\Big) + \alpha\|\mathbf{w}\|_{\mathbf{M}}^2. \quad (4.88) \end{aligned}$$

Because $\|\mathbf{w}\|_{\mathbf{M}}^2 = \mathbf{y}^T\mathbf{\Psi}^T\mathbf{M}\mathbf{\Psi}\mathbf{y} = \mathbf{y}^T\mathbf{I}_{n-k}\mathbf{y} = \|\mathbf{y}\|_2^2$ we conclude that α forms a lower bound for the eigenvalues of the Schur complement $\mathbf{S}$. An upper bound is found by letting $\mathbf{v} = \mathbf{0}$ and choosing an arbitrary $\mathbf{y} \in \mathbb{R}^{n-k}$ of unit size, i.e. $\|\mathbf{y}\| = 1$,

$$\begin{aligned} \mathbf{y}^T\mathbf{S}\mathbf{y} &= \inf_{\mathbf{v}\in\mathcal{V}_k} \left[\|\mathbf{K}(\mathbf{v} + \mathbf{w})\|_2^2 + \alpha\|\mathbf{v}\|_{\mathbf{M}}^2\right] + \alpha\|\mathbf{w}\|_{\mathbf{M}}^2 \\ &\le \|\mathbf{K}\mathbf{w}\|_2^2 + \alpha\|\mathbf{w}\|_{\mathbf{M}}^2 \\ &\le \gamma_{\mathbf{\Psi}}^2 + \alpha. \quad (4.89) \end{aligned}$$

The lower and upper eigenvalue bound yields the condition number estimate

$$\mathrm{cond}(\mathbf{S}) \le 1 + \frac{\gamma_{\mathbf{\Psi}}^2}{\alpha}. \quad (4.90)$$

Already at this point we see a significant advantage to the Schur complement because the α in the nominator is not squared as in (4.86). From Axelsson [1, Lemma 3.12] we also know that the Schur complement is guaranteed to have a lower condition number than the system it is constructed from.

4.4.4 Symmetric Gauss-Seidel Condition Number

The condition number of the Gauss-Seidel preconditioned system is strongly related to the equivalent of the Schur complement.

The symmetric Gauss-Seidel preconditioner can be reformulated into a block Cholesky factorization

$$\begin{aligned} \mathbf{N}_{GS} &= (\widehat{\mathbf{D}} + \mathbf{L})\widehat{\mathbf{D}}^{-1}(\widehat{\mathbf{D}} + \mathbf{U}) \\ &= (\widehat{\mathbf{D}} + \mathbf{L})\widehat{\mathbf{D}}^{-1/2}\widehat{\mathbf{D}}^{-1/2}(\widehat{\mathbf{D}} + \mathbf{L})^T \\ &= \mathbf{C}\mathbf{C}^T, \end{aligned}$$

where $\mathbf{C} = (\widehat{\mathbf{D}} + \mathbf{L})\widehat{\mathbf{D}}^{-1/2}$.

Combined with the block representation of $\mathbf{A}$ we get (cf. (3.21)) a matrix with the same eigenvalues as the preconditioned system

$$C^{-1}\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix} C^{-T}$$
$$=$$
$$\begin{bmatrix} \mathbf{A}_{11}^{-1/2} & \mathbf{0}_{k,n-k} \\ -1/\sqrt{\alpha}\mathbf{A}_{21}\mathbf{A}_{11}^{-1} & 1/\sqrt{\alpha}\mathbf{I}_{n-k} \end{bmatrix}\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix}\begin{bmatrix} \mathbf{A}_{11}^{-1/2} & -1/\sqrt{\alpha}\mathbf{A}_{11}^{-1}\mathbf{A}_{12} \\ \mathbf{0}_{n-k,k} & 1/\sqrt{\alpha}\mathbf{I}_{n-k} \end{bmatrix}$$
$$=$$
$$\begin{bmatrix} \mathbf{I}_k & \mathbf{0}_{k,n-k} \\ \mathbf{0}_{n-k,k} & \mathbf{S}/\alpha \end{bmatrix},$$

where $\mathbf{S}$ is the usual Schur complement (4.36). The preconditioned system has the eigenvalues of $\mathbf{S}$ divided by α plus a k-multiple eigenvalue 1. If

1 lies in the eigenvalue spectrum of $\mathbf{S}/\alpha$ we will have the same condition number as for the Schur complement — in all other cases we will have a larger condition number. From [1, Lemma 3.12] we have that

$$\begin{array}{rcl} \lambda_{\min}(\mathbf{S}) & \geq & \lambda_{\min}(\mathbf{A}) = \alpha + \sigma_n^2 \\ & \Downarrow & \\ \lambda_{\min}(\mathbf{S}/\alpha) & \geq & 1 + \sigma_n^2/\alpha, \end{array}$$

which implies that 1 is outside the eigenvalue spectrum of $\mathbf{S}$ — but still very close in the usual case $\alpha > \sigma_n^2$. The condition numbers of the Schur complement and the Gauss-Seidel system is therefore very close to each other.

Furthermore we know that clustered eigenvalues have a positive effect on CG and the k extra eigenvalues can not be more clustered than they are.

Summary on Condition Numbers

We have now derived upper bounds for the condition numbers of the three methods. The results are summed in Table 4.2 and in Chapter 6 we will see if the convergence behaves as predicted by these estimations.

Method	Condition Number
Jacobi	$(\gamma_{\Psi}^2 \sigma_1^2)/\alpha^2$
Gauss-Seidel	$1 + \gamma_{\Psi}^2/\alpha$
Schur Complement	$1 + \gamma_{\Psi}^2/\alpha$
No preconditioning	σ_1/α

Table 4.2: *Estimated condition numbers for each method. The regularization parameter α is assumed to be small but larger than σ_n^2.*

4.5 Final Remarks

All the previous methods work on the equation system $\mathbf{K}^T\mathbf{K}+\alpha\mathbf{L}^T\mathbf{L}$. The preconditioners are constructed from this system and work on this system. We have tried the variation of letting the preconditioner be based on the regularized system but the actual system be the unregularized $\mathbf{K}^T\mathbf{K}$.

However this renders the estimates derived in this chapter useless, but a practical approach is still possible. Section 6.9.3 has results on this variation.

Chapter 5

Subspaces

Heavy use of the abstract notions of vector space and subspace in a discussion of numerical methods may seem to some readers unnecessarily abstruse. In fact, however, the language of subspaces simplifies such discussions by suppressing distracting details. Parlett [25]

We have until now only scratched the surface of how to select the actual subspace splitting. In this chapter we discuss some choices of subspaces and expectations to their performance.

5.1 A Good Subspace

The discussion of convergence properties, in the previous chapter, showed that a subspace created from the right singular vectors of the SVD is optimal, because $\gamma_{\mathbf{\Psi}}$ is minimized. Furthermore we get a diagonal system which is easy to solve. However, we do not have the SVD — if we had, the problem would not be a problem and an iterative method would be utterly unnecessary.

The hope is that a subspace close to the SVD subspace will do almost as good. In the treatment of ill-posed problems it was mentioned that the singular vectors often have an increasing number of zero-crossings as the index increases. This observation will be used as a starting point.

5.1.1 Cosine and Sine

The most obvious choice of a more and more oscillatory basis must be a basis created by sine and cosine functions with increasing frequency. This choice implies that $[\mathbf{\Phi}\ \ \mathbf{\Psi}]$ turns into a discrete Fourier transform (DFT) and the transposed $[\mathbf{\Phi}\ \ \mathbf{\Psi}]^T$ takes the form of the inverse discrete Fourier transform (IDFT).

An added bonus is that the projection to and from the subspace $\mathcal{V}_k$ can be obtained efficiently from the well known fast Fourier transform (FFT). In the experiments section we have not used the FFT approach but the "naive" matrix-vector multiplication approach. The complexity of a FFT is $\mathcal{O}(n \log n)$ while a matrix-vector transformation takes $2kn$ flops and the choice between the two methods depends on the exact hidden coefficient in the $\mathcal{O}$-notation (which changes from implementation to implementation) and the subspace size k. The FFT implementation in Matlab assumes n to be a power of two although algorithms for other n exist [21] and is therefore not suited for a general implementation of our methods.

5.1.2 Chebyshev Polynomials

During the discussion of CG convergence Chebyshev polynomials were used because their absolute value is less or equal to one in the interval $[-1; 1]$. The Chebyshev polynomials reappear in this chapter because they also have increasingly many zero-crossings as the order of the polynomial increases. One way of calculating the kth Chebyshev polynomial is

$$P_k(x) = \cos\left(k \operatorname{acos}(x)\right).$$

We note that a Chebyshev polynomial can be viewed as a cosine function with an abscissa stretched such that the function appears as a polynomial. Hence we expect a basis constructed from Chebyshev polynomials to do similar to a cosine/sine basis.

5.1.3 Wavelets

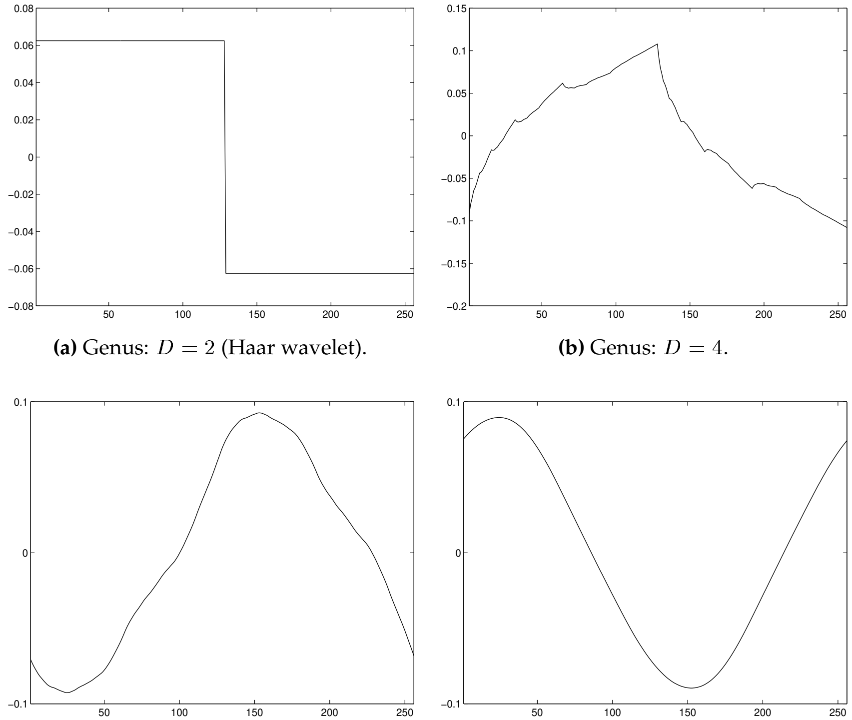
A cousin of the Fourier transform is the wavelet transform [4] [23] which has many similar properties but also a few differences compared with the

Fourier transform. A wavelet basis is described by two functions (the scaling and the mother wavelet function), and a basis is obtained by translating and resizing these functions. A wavelet can have "local" oscillation whereas a sine or cosine oscillate throughout the entire interval. Many families of wavelets exist but we have only considered the well known Daubechies family.

A nice similarity to the cosine/sine basis is the possibility of a fast transforms, the fast (inverse) wavelet transform FWT and IFWT. The flop count for a fast (inverse) wavelet transform is $4Dn$ where D, the genus, is an even integer characterizing the length of a filter. The (I)FWT requires n to be a power of 2, but unlike FFT it is unclear whether fast transforms exist for other problem sizes. The package made by Ole Møller Nielsen (OMN) [23] contains code to make Daubechies wavelets of genus 2 to 50. A visual inspection, see Figure 5.1 for some examples, shows that the wavelets look more like a piece of sine or cosine when the genus is large, while a lower genus gives less smooth functions. Hence we expect that a large genus will yield better results than a smaller genus, but considering the added costs a reasonable tradeoff should be chosen. Whether we actually gain good results (and for which genus) in terms of convergence etc. are the topic of Section 6.7.

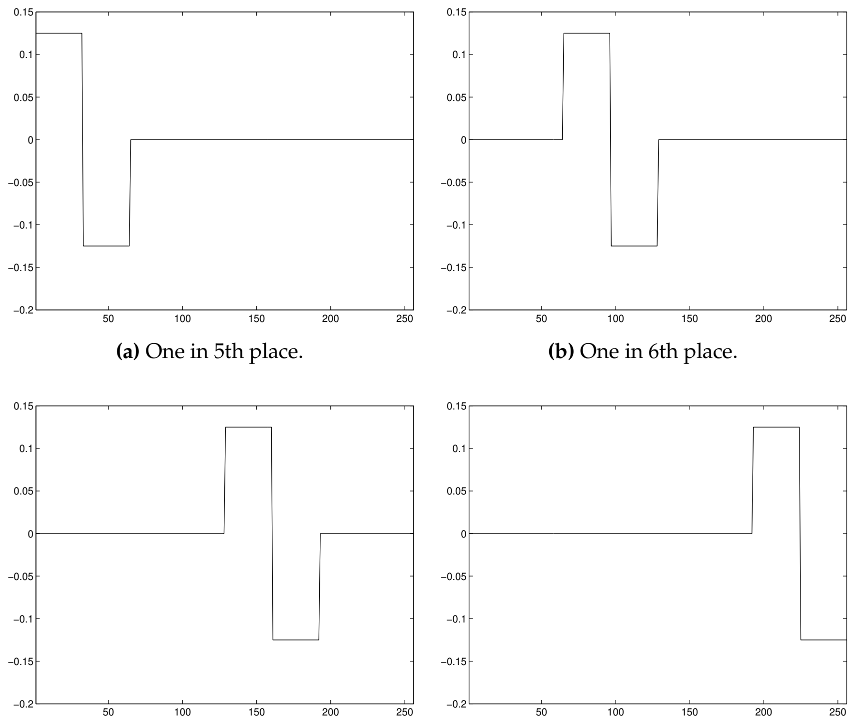
The wavelets in Figure 5.1 are all the the "mother wavelet" ($\psi_0(x)$ in OMN thesis) constructed by a wavelet transform of a vector with zeros in all places except for a one in the second. We now freeze the Genus ($D = 2$) and plot the fifth to the eight wavelet ($\psi_{2,0}(x)$ to $\psi_{2,3}(x)$). Figure 5.2 shows that all four wavelets are necessary to hold information over the hole interval. The next "batch" of wavelets holds 8 wavelets because they have half the width ($\psi_{3,0}(x)$ to $\psi_{3,7}(x)$). The more "batches" we include the finer details can be approximated, but incomplete batches are likely to be unable to approximate in certain parts of the given interval. Hence we suspect that a wavelet basis should be chosen such that the dimension is 2, 4, 8 or another power of two.

In order to test the efficiency of the wavelet transform a special version of Schur complement CG has been developed that utilizes OMN's package, see Appendix A.5.1.



(a) Genus: $D = 2$ (Haar wavelet). **(b)** Genus: $D = 4$. **(c)** Genus: $D = 8$. **(d)** Genus: $D = 16$.

Figure 5.1: *Shape of Daubechies wavelets (the mother wavelet) with different genus. A high genus yields a "smooth" wavelet but oscillations increase. The downside is a more expensive fast transform.*



(a) One in 5th place.

(b) One in 6th place.

(c) One in 7th place.

(d) One in 8th place.

Figure 5.2: *Shapes of Daubechies ($D = 2$) wavelets at same scale. Multiple wavelets are necessary to describe information on the hole interval.*

5.1.4 Lanczos Vectors

The Wavelet and Fourier choices for subspace transformations adhere to the idea of imitating the normal behaviour of the singular vectors. The Lanczos vectors [7] are a collection of orthogonal vectors that span the Krylov space made from the problem in question. These vectors are known to approximate the vectors of the largest eigenvalues. Hence it is expected that a subspace spanned by Lanczos vectors is a good choice. The use of Lanczos vectors gives birth to an "bootstrap" idea because the Lanczos vectors are easily extracted from the CG-algorithm. The idea is to incrementally increase the subspace $\mathcal{V}_k$ as the iterations are performed and the Lanczos vectors are generated. This idea seems promising until one considers two factors:

1. CG minimizes the error $\|\mathbf{e}_{(k)}\|_{\mathbf{A}}$ in the Krylov space $\mathcal{K}(\mathbf{A}, \mathbf{b}, k)$ (assuming infinite precession) at each step.
2. The Lanczos vectors span the exact same subspace as the Krylov space $\mathcal{K}(\mathbf{A}, \mathbf{b}, k)$. If the columns of $\mathbf{\Phi}$ span the very same subspace $\mathcal{K}(\mathbf{A}, \mathbf{b}, k)$ then $\mathbf{\Phi}\mathbf{A}_{11}^{-1}\mathbf{\Phi}^T\mathbf{r}$ computes a correction that minimizes the error, restriced to the Krylow space, but this time in the 2-norm.

If the $\mathbf{A}$-norm and 2-norm are almost equal preconditioning will not improve results greatly. However, this thought is examined in Section 6.7, where an experiment shows that the Lanczos vectors are a good choice for the particular test problem despite the arguments expressed here. Hence it it could prove valuable to follow the bootstrap idea further.

5.1.5 Simulated Singular Vectors (regutm)

The REGULARIZATION TOOLS package [17] includes a function to create random test problems where the singular vectors behave as if they were from an ill-posed problem. Among the return values are the singular vectors of this random problem which then could be used on a different problem. This approach should only be used for small problems as it involves calculating an SVD.

5.2 Two Dimensional Deconvolution

The previous subspaces do not suit a two dimensional problem very well. In this section, an adaption of [14, Chapter 6], we take a look at the two-dimensional convolution problem

$$\int_{I_1}\int_{I_2} K(x,y,\hat{x},\hat{y})d\hat{x}d\hat{y} = g(x,y), \tag{5.1}$$

where the kernel K has the form

$$K(x,y,\hat{x},\hat{y}) = \kappa(x-\hat{x})\omega(y-\hat{y}), \tag{5.2}$$

where κ and ω are functions (the variables separate).

This problem can, when discretized, be formulated using a Kronecker product. A Kronecker product of two matrices $\mathbf{A} \in \mathbb{R}^{m\times n}$ and $\widetilde{\mathbf{A}} \in \mathbb{R}^{\widetilde{m}\times\widetilde{n}}$ is defined as

$$\mathbf{A}\otimes\widetilde{\mathbf{A}} = \begin{pmatrix} a_{1,1}\widetilde{\mathbf{A}} & a_{1,2}\widetilde{\mathbf{A}} & \cdots & a_{1,n}\widetilde{\mathbf{A}} \\ a_{2,1}\widetilde{\mathbf{A}} & a_{2,2}\widetilde{\mathbf{A}} & \cdots & a_{2,n}\widetilde{\mathbf{A}} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1}\widetilde{\mathbf{A}} & a_{m,2}\widetilde{\mathbf{A}} & \cdots & a_{m,n}\widetilde{\mathbf{A}} \end{pmatrix} \in \mathbb{R}^{(m\widetilde{m})\times(n\widetilde{n})}. \tag{5.3}$$

The usefulness of the Kronecker product is apparent if we consider some of the important algebraic rules for the Kronecker product

$$\begin{aligned} (\mathbf{A}\otimes\mathbf{B})^T &= \mathbf{A}^T\otimes\mathbf{B}^T, && (5.4)\\ (\mathbf{A}\otimes\mathbf{B})^{-1} &= \mathbf{A}^{-1}\otimes\mathbf{B}^{-1}, && (5.5)\\ (\mathbf{A}\otimes\mathbf{B})\mathrm{vec}(\mathbf{X}) &= \mathrm{vec}(\mathbf{B}^T\mathbf{X}\mathbf{A}^T), && (5.6)\\ (\mathbf{A}\otimes\mathbf{B})(\mathbf{C}\otimes\mathbf{D}) &= (\mathbf{AC})\otimes(\mathbf{BD}), && (5.7) \end{aligned}$$

where the function $\mathrm{vec}(\cdot)$ stacks the columns of the argument matrix. Equation (5.6) can give great savings in terms of both calculations as well as memory. Assume that $\mathbf{A} = \mathbf{B} \in \mathbb{R}^{n\times n}$. Then a naive multiplication $(\mathbf{A}\otimes\mathbf{B})\mathrm{vec}(\mathbf{X})$ takes $2n^4$ flops (assuming that $\mathbf{A}\otimes\mathbf{B}$ is already calculated), while the calculation $\mathbf{AXB}$ only uses $4n^3$ flops — a great saving. In addition the Kronecker product matrix is *much* larger than the two creators.

All previous work has been working on one-dimensional data, and in order to treat two-dimensional data in form of a matrix we must reshape the data into a vector. The standard procedure is to stack the 2-D data columnwise. Assuming we have the discretizations $\mathbf{K}_1$ and $\mathbf{K}_2$ of κ and ω respectively, we are able to describe the convolution using a Kronecker product

$$(\mathbf{K}_1\otimes\mathbf{K}_2)\mathrm{vec}(\mathbf{X}) = \mathrm{vec}(\mathbf{B}),$$

and using (5.6) we get

$$\mathbf{K}_2\mathbf{X}\mathbf{K}_1^T = \mathbf{B}. \tag{5.8}$$

Secondly we have to realize how the singular vectors of the Kronecker product are connected to the singular vectors of the two kernels. A simple calculation involving the SVD of $\mathbf{K}_1 = \mathbf{U}_1\mathbf{\Sigma}_1\mathbf{V}_1^T$ and $\mathbf{K}_2 = \mathbf{U}_1\mathbf{\Sigma}_1\mathbf{V}_2^T$ and (5.7) proceeds as follows

$$\begin{aligned} \mathbf{K}_1\otimes\mathbf{K}_2 &= (\mathbf{U}_1\mathbf{\Sigma}_1\mathbf{V}_1^T)\otimes(\mathbf{U}_2\mathbf{\Sigma}_2\mathbf{V}_2^T)\\ &= (\mathbf{U}_1\otimes\mathbf{U}_2)\left((\mathbf{\Sigma}_1\mathbf{V}_1^T)\otimes(\mathbf{\Sigma}_2\mathbf{V}_2^T)\right)\\ &= (\mathbf{U}_1\otimes\mathbf{U}_2)(\mathbf{\Sigma}_1\otimes\mathbf{\Sigma}_2)(\mathbf{V}_1^T\otimes\mathbf{V}_2^T). && (5.9) \end{aligned}$$

Equation (5.9) reveals that the singular vectors of a Kronecker product themselves are the result of a Kronecker product. Hence a subspace generated by means of a Kronecker product of two suitable 1-D singular vector approximations as those previously described could prove useful.

The application of the convolution operator is implemented efficiently using (5.8), but we have not exploited the Kronecker product to its fullest yet. The transformation to and from $\mathcal{V}_k$ is easy when $\mathbf{\Phi} = \mathbf{\Phi}_1\otimes\mathbf{\Phi}_2$. The transformation $\mathbf{y} = \mathbf{\Phi}^T\mathbf{x}$ is

$$\begin{aligned} \mathrm{vec}(\mathbf{Y}) &= (\mathbf{\Phi}_1\otimes\mathbf{\Phi}_2)^T\,\mathrm{vec}(\mathbf{X})\\ &\Updownarrow\\ \mathbf{Y} &= \mathbf{\Phi}_2^T\mathbf{X}\mathbf{\Phi}_1, \end{aligned}$$

and the "reverse operation" is

$$\begin{aligned} \mathrm{vec}(\mathbf{X}) &= (\mathbf{\Phi}_1\otimes\mathbf{\Phi}_2)\,\mathrm{vec}(\mathbf{Y})\\ &\Updownarrow\\ \mathbf{X} &= \mathbf{\Phi}_2\mathbf{Y}\mathbf{\Phi}_1^T. \end{aligned}$$

Finally the creation of $\mathbf{A}_{11}$ goes without any surprises

$$\begin{aligned}
\mathbf{A}_{11} &= \mathbf{\Phi}^T\mathbf{K}^T\mathbf{K}\mathbf{\Phi} + \alpha\mathbf{\Phi}^T\mathbf{L}^T\mathbf{L}\mathbf{\Phi} \\
&= (\mathbf{\Phi}_1^T \otimes \mathbf{\Phi}_2^T)(\mathbf{K}_1^T \otimes \mathbf{K}_2^T)(\mathbf{K}_1 \otimes \mathbf{K}_2)(\mathbf{\Phi}_1 \otimes \mathbf{\Phi}_2) + \alpha\mathbf{\Phi}^T\mathbf{L}^T\mathbf{L}\mathbf{\Phi} \\
&= (\mathbf{\Phi}_1^T\mathbf{K}_1^T\mathbf{K}_1\mathbf{\Phi}_1) \otimes (\mathbf{\Phi}_2^T\mathbf{K}_2^T\mathbf{K}_2\mathbf{\Phi}_2) + \alpha\mathbf{\Phi}^T\mathbf{L}^T\mathbf{L}\mathbf{\Phi} \qquad (5.10)
\end{aligned}$$

However, even if we require $\mathbf{L}$ to be a Kronecker product there is no easy way of inverting (5.10) because of the addition. Hence it is necessary to compute $\mathbf{A}_{11}$ explicitly.

A possible work-around is to forget $\alpha\mathbf{\Phi}\mathbf{L}^T\mathbf{L}\mathbf{\Phi}$ and just use the first term $\widehat{\mathbf{A}}_{11} = (\mathbf{\Phi}_1^T\mathbf{K}_1^T\mathbf{K}_1\mathbf{\Phi}_1) \otimes (\mathbf{\Phi}_2^T\mathbf{K}_2^T\mathbf{K}_2\mathbf{\Phi}_2)$. If the subspace is selected correctly, that is from the SVD of the kernels, the same arguments used to delete $\mathbf{\Psi}^T\mathbf{K}^T\mathbf{K}\mathbf{\Psi}$ from $\mathbf{A}_{22}$ (Section 4.2.1) can be used again, albeit "reversed", because we now hope for $\sigma_k^2 > \alpha$. With this simplification and (5.5) we get

$$\mathbf{A}_{11}^{-1}\mathrm{vec}(\mathbf{X}) \approx \widehat{\mathbf{A}}_{11}^{-1}\mathrm{vec}(\mathbf{X}) = \mathrm{vec}\left((\mathbf{\Phi}_2^T\mathbf{K}_2^T\mathbf{K}_2\mathbf{\Phi}_2)^{-1}\mathbf{X}(\mathbf{\Phi}_1^T\mathbf{K}_1^T\mathbf{K}_1\mathbf{\Phi}_1)^{-1}\right)$$

The approximation gives savings in both the initialization phase and in each iteration. Assume that k is the number of columns of both $\mathbf{\Phi}_1$ and $\mathbf{\Phi}_2$. Then the cost of a factorization of the Kronecker factors is $\mathcal{O}(k^3)$, while a factorization of the Kronecker product has the complexity $\mathcal{O}(k^6)$. When solving a system with one of the Kronecker factors the cost is $\mathcal{O}(k^2)$, while the cost of solving a system with the Kronecker product is $\mathcal{O}(k^4)$.

Appendix A.6.1 shows an implementation of a Schur complement CG suited for 2-D problems of the type just presented and in Section 6.9.1 we present the result of a simple experiment including an example on whether the discussed simplification has any noticeable influence.

N-Dimensional Problems

We have taken the first step towards n-dimensional problems. It does not take much fantasy to expand the 2-D techniques to 3 dimensions. However we need a new type of matrix with 3 dimensions that can be multiplied from the usual left and right but also from above (or below). This extension or generalization of matrices is known as tensors — a subject outside the scope of this thesis. The test problems include one 3-D problem but we do not utilize the obvious extension of the Kronecker product rules to 3 dimensions. Instead we use the usual matrix-vector approach.

Chapter 6

Numerical Experiments

"There are three principal means of acquiring knowledge (...) observation of nature, reflection, and experimentation. Observation collects facts; reflection combines them; experimentation verifies the result of that combination." Denis Diderot, French philosopher (1713-1784).

Numerical experiments with the previously described algorithms and their parameters are presented in the following. The methods are compared to the standard CGLS from REGULARIZATION TOOLS [17].

With the many parameters, test problems etc., it is impossible to cover all aspects. With this chapter we have covered what we find most important and relevant. In the following we have abbreviated the Gauss-Seidel precondition equation system to GS and Schur complement CG is shortened Schur CG. Any reference to a subspace or a basis refer to the the subspace or basis spanned by the columns of $\mathbf{\Phi}$. Furthermore a SVD basis denotes a basis derived from the kernel of the problem of current interest. Other bases refer to Chapter 5. We have unless otherwise stated used the regularization matrix $\mathbf{L} = \mathbf{I}_n$.

6.1 The Test Problems

We have selected a number of test problems in order to test and compare the presented algorithms. Three problems are taken from REGULARIZATION TOOLS and they possess the role of the small problems. The selected

problems all have the property that the singular values do not decay fast towards zero, i.e., they do not have ten singular values above machine precision and the rest around the machine precision. Thus CG converges slowly which gives a preconditioner a possibility to improve the situation. To complement the REGULARIZATION TOOLS problems, two "real world" problems, both of considerable size, are treated. Now follows a short description of the problems.

heat: Inverse heat transfer

This particular 1-dimensional problem is taken from [17]. It is a discretization of a Volterra integral equation of the first kind. Hence the kernel is $K(s,t) = k(s-t)$, where

$$k(t) = \frac{t^{-3/2}}{2\kappa\sqrt{\pi}} \exp\left(-\frac{1}{4\kappa^2 t^2}\right).$$

The constant κ describes the heat conducting capabilities of the material observed. A large constant ($\kappa \approx 5$) yields a well-conditioned matrix, while a small ($\kappa \approx 1$) gives an ill-conditioned matrix. We have in all experiments used $\kappa = 1$ unless otherwise stated.

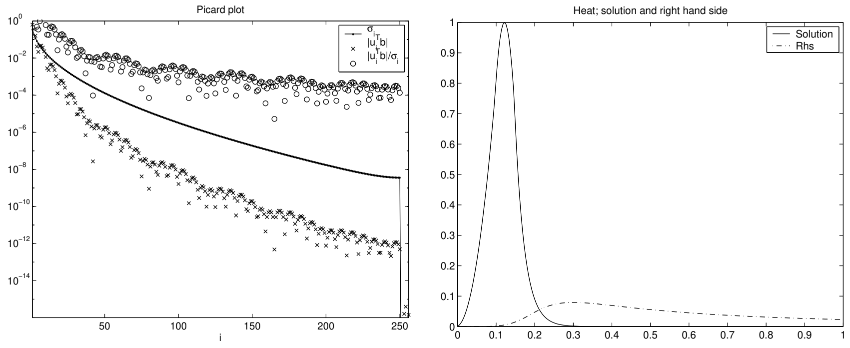
Figure 6.1 shows the Picard plot and a plot of the solution and the corresponding right-hand side. Note that the singular values decay slowly and we must expect a slow convergence from an unpreconditioned CG method. This makes heat our primary test problem. In addition we note an drop of the singular values at the end — a sign of rank deficiency. However, the effective condition number is approximately 10^8.

deriv2: Computation of the Second Derivative

Also from [17] we have picked deriv2. This test problem is based on a discretization of a Fredholm integral equation with the kernel

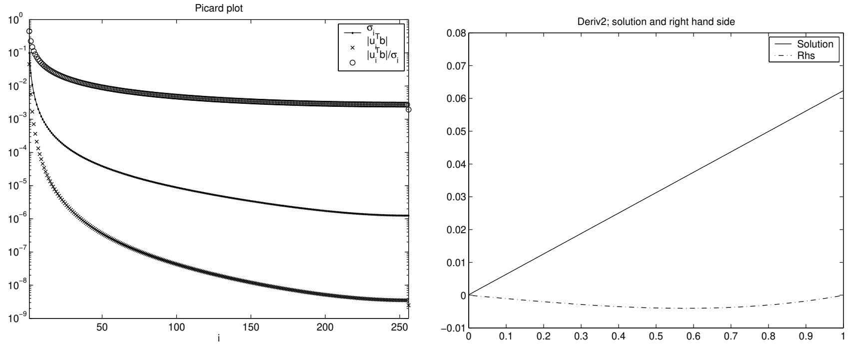
$$K(s,t) = \begin{cases} s(t-1) & s < t \\ t(s-1) & s \geq t \end{cases}.$$

Figure 6.2 shows the Picard plot and a plot of the solution and right-hand side. Observe that the singular values decay slower than those of heat and



(a) Picard plot. **(b)** Solution and right-hand side.

Figure 6.1: *Profile of test problem* heat. *The Picard plot shows a slow decay of the singular values and a drop in the very end, where $\sigma_{251}, \ldots, \sigma_{256} \ll 10^{-16}$, i.e., the problem is rank deficient.*



(a) Picard plot. **(b)** Solution and right-hand side.

Figure 6.2: *Profile of test problem* deriv2. *The Picard plot shows a slow and smooth decay of the singular values.*

that we do not have rank deficiency. The condition number is approximately 10^5, i.e., less than the effective condition number of heat. Therefore we expect faster convergence for deriv2.

blur: Inverse atmospheric turbulence blur

This test problem is also from [17] and it is our first 2-dimensional problem. The system models the degradation of images by atmospheric turbulence blur. To be more specific a Gaussian point-spread function is used

$$h(\Delta x, \Delta y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{\Delta x^2 + \Delta y^2}{2\sigma^2}\right),$$

which describes how the energy at a pixel is smeared. A large σ spreads the energy more than a small σ. Thus a large σ yields a more ill-conditioned system.

This problem is selected because it is constructed using a Kronecker product of a kernel $\mathbf{K}$ with itself. Therefore we can use the approach from Section 5.2 with the possibility of solving much larger problems.

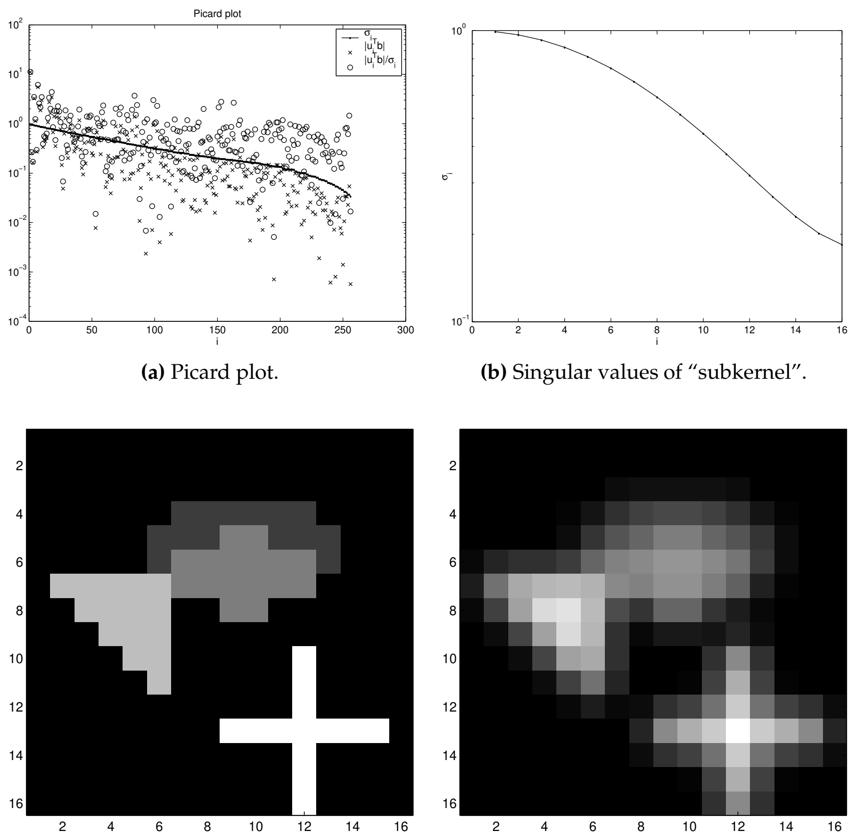
Figure 6.3 shows the Picard plot, the sharp image (the solution) and the blurred image (the right-hand side) of a small problem. Furthermore we see a plot of the singular values of the kernel $\mathbf{K}$.

geomig: Geophysical Migration

This test problem is kindly provided by Yann-Hervé De Roeck. The problem is described in detail in [30]. Basicly the problem is to determine where layer boundaries are situated in the soil — very usefull when searching for oil.

The problem used for investigation has a matrix $\mathbf{K} \in \mathbb{R}^{120000 \times 8286}$ with approximately $7.8 \cdot 10^6$ non-zeros[1]. The matrix acts on a vector originating from a 101×95 grid stacked columnwise. That is we have a two dimensional problem.

[1] Using a convolution with a signal of length 3, i.e., the problem is in difficulty between problem 1 (length 1) and problem 1" (length 9) of [30].

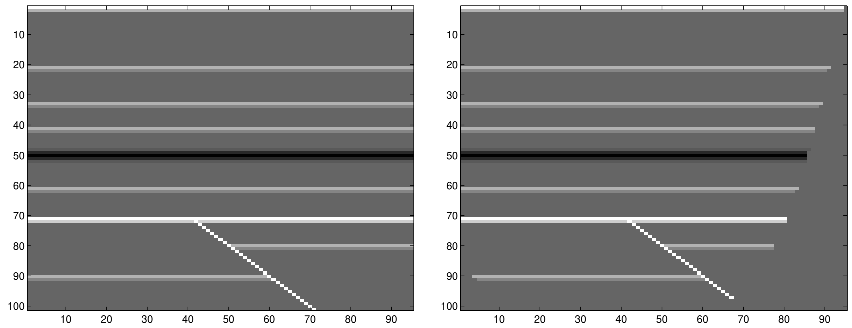


(a) Picard plot. **(b)** Singular values of “subkernel”.

(c) The sharp unblurred solution. **(d)** The blurred right-hand side.

Figure 6.3: *Profile of test problem* blur*. The Picard plot shows an well-conditioned system with* $\mathrm{cond}(\mathbf{K} \otimes \mathbf{K}) \approx 30$*. In this example we have used* $\sigma = 0.7$*.*

The right-hand side is generated from a simulated solution, see Figure 6.4(a), and the provided matrix. The geometry of the physical model yields zero columns in the system matrix which means that some elements are not recoverable through inversion. Figure 6.4(b) shows the solution with the unrecoverable parts deleted.



(a) Reflectivity. **(b)** Reflectivity possible to reconstruct.

Figure 6.4: *Solution to the geophysical migration problem.*

vesuvio: Inversion of Geomagnetic Data

The final test problem is provided by Valeria Paoletti. The problem models the connection between the magnetic field below the earth surface and measurements taken above the surface. The solution domain is a cube (below the surface) discretized with grid of size $10 \times 10 \times 10$, while the measurements domain is a grid of size $10 \times 10 \times 10$ (corresponding to measurements in 10 layers above the surface). The results is a dense system matrix $\mathbf{K} \in \mathbb{R}^{1000 \times 1000}$. The equation system is “almost” a result of a Kronecker product, that is the mathematics tells us that it is not a Kronecker product but an estimation of the SVD shows very Kronecker like properties. The problem is described in detail in [6].

We have used simulated data for this experiment and the right-hand side is generated using a multiplication with the system matrix and the simulated solution. The solution is a $3 \times 3 \times 4$ block placed 3 units below the surface. This particular placement is chosen because information below the top most layer is known to be hard to reconstruct with the standard CGLS method.

6.2 Initial Comparison

The theoretical condition number bounds from Table 4.2 suggest that the Jacobi preconditioner is inferior to GS and Schur CG. We have tested this hypothesis on the test problem heat.

6.2.1 Condition Numbers

Even though the condition number does not tell the entire truth about the convergence it still carries information on what to expect — and it is easier to estimate than all eigenvalues. Furthermore the results are only influenced by the system matrix and not by the solution or the right-hand side. Hence we are able to conclude on a more general level. The actual convergence depends on both the singular values and the right-hand side coefficients, cf. Section 3.2.

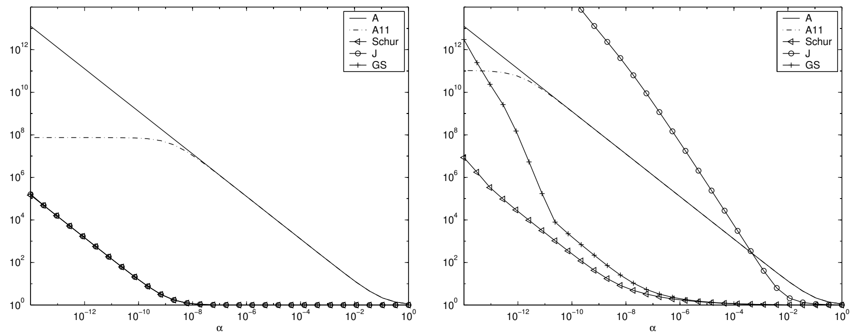
We concentrate on the influence of α and the subspace dimension on the condition number of the preconditioned systems. We have chosen heat because the condition number of the unpreconditioned and unregularized system is huge. Thus we will see an effect of applying regularization even for very small α. We only compare the optimal SVD basis with the regutm basis. Similar comparisons of other subspace selections are postponed to Section 6.7.

Varying α

The variation of α influences both the regularized system and the preconditioner. Figure 6.5 shows the condition number as a function of α using an SVD basis and a regutm basis. The condition numbers are equal for the Jacobi and GS method if an SVD basis is used. Furthermore we note an improved condition number compared to the unpreconditioned system. However, turning to the suboptimal subspace, we see that the Jacobi preconditioner is very sensitive to the selected subspace and becomes *worse*

than the unpreconditioned system. Both GS and Schur have higher condition numbers compared to those obtained with the SVD basis, but they are still better conditioned than the unpreconditioned system. The condition number of $\mathbf{A}_{11}$ is included because its inverse is used in all iterations. Because $\sigma_{64}^2 \approx 2 \cdot 10^{-9}$ (64 is the subspace dimension) the condition number of $\mathbf{A}_{11}$ is only influenced when $\alpha \geq 10^{-9}$, where the regularization also influences the larger singular values. In addition the lower singular values have been "eliminated" yielding condition numbers of approximately 1 for Jacobi, GS and Schur CG.

If a suboptimal basis is used we the influence of α on $\mathbf{A}_{11}$ is visible for smaller α and the initial level (that is for very small α) the condition number is larger. The suboptimal subspace in this case also includes components from right singular vectors corresponding to smaller singular values, that is $\sigma_{65}, \ldots, \sigma_n$. Thus we get an effect on the condition number of $\mathbf{A}_{11}$ also for $\alpha < \sigma_{64}^2$. Furthermore the complementary subspace spanned by $\mathbf{\Psi}$ now also spans components from larger singular values and the condition numbers of GS and Schur go slower to zero as the regularization parameter α increases.



(a) SVD basis. Jacobi condition number is hidden behind the GS condition number.

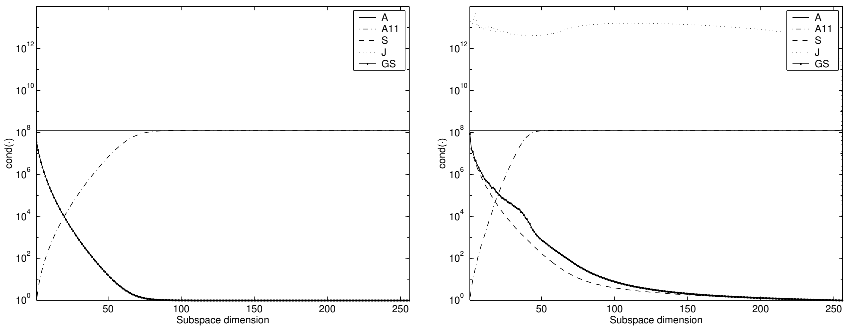
(b) regutm basis.

Figure 6.5: *Condition number as a function of α. The subspace dimension is locked at 64.*

Subspace Dimension

We know the condition number of preconditioned systems depends on $\gamma_{\Psi} = \|\mathbf{K}|_{\mathcal{W}_k}\|_{\mathbf{M}}$, which in turn depends on the subspace dimension. We now freeze α and change the dimension of the coarse subspace $\mathcal{V}_k$.

Figure 6.6 shows the condition number changes as more and more vectors are used for the coarse subspace. We see the obvious result that the condition number of the the preconditioned systems and the Schur complement all are 1 when $\mathcal{V}_k$ spans the entire space. But at the same time $\mathbf{A}_{11}$ have the condition number of the unpreconditioned system, because it *is* the unpreconditioned system (modified by a similarity transformation). At the other end a very small subspace (1–10 dimensions) does not gain much and seems in this case that a subspace dimension of, e.g., 10 would be appropriate, because the condition of $\mathbf{A}_{11}$ is still limited. If we compare the two types of bases we see the SVD basis performs better than the basis extracted from regutm. Especially the Jacobi preconditioner suffers from the non-SVD subspace and the condition number is worse than that of the unpreconditioned system — even for large subspace dimensions. Finally we observe that the Schur complement is slightly better conditioned than GS. These practical observations comply with the theoretical observations in Section 4.4.



(a) SVD basis. Schur, GS and Jacobi are equal.

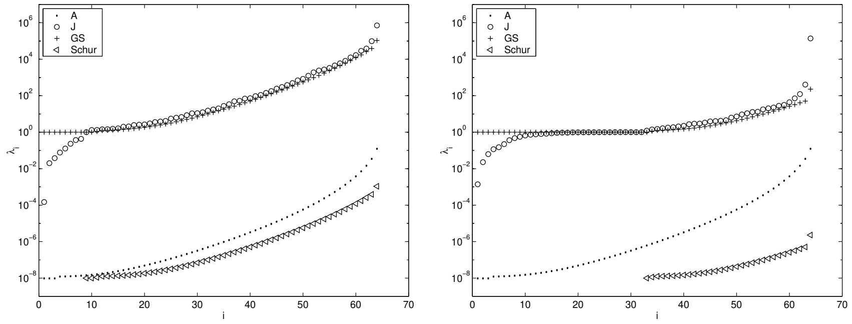
(b) regutm basis.

Figure 6.6: *Condition number as a function of subspace dimension. Test problem* heat *with* $\alpha = 10^{-9}$ *implying* $\mathrm{cond}(\mathbf{A}) \approx 3 \cdot 10^8$. *Notice that Jacobi preconditioning is very sensitive to the subspace selection.*

6.3 Eigenvalue Distribution

We have now seen how the two extreme eigenvalues behave for our methods. How do the non-extreme eigenvalues change when preconditioning is applied?

Figure 6.7 shows the distribution of eigenvalues of the regularized system and of the two preconditioned systems. GS and Jacobi "share" many eigenvalues but at the ends Jacobi has a couple "running off". Hence the larger condition number of the Jacobi preconditioned system. GS does not have these and is furthermore better conditioned than the unpreconditioned system. Comparing the left and right plot we see that a higher dimensional subspace creates more eigenvalues $\lambda = 1$ for GS, i.e., higher clustering, in addition to the better conditioning. This complies with the fact that GS should have the exact same eigenvalues as Schur CG (divided by α) plus a k-multiple eigenvalue 1. Tables 6.1 and 6.2 display the largest and smallest eigenvalues of the considered systems and the tables confirm the predictions from Section 4.4, that is the condition number of the Jacobi system is approximately that of GS or Schur squared.



(a) 8-D subspace.

(b) 32-D subspace.

Figure 6.7: *Eigenvalues for regularized system and preconditioned regularized systems. Regularization parameter* $\alpha = 10^{-8}$ *and subspace generation by* regutm. *The problem is* heat *with* $n = 64$. *The eigenvalues of the Schur system is shifted to the right such that the similarities to GS become apparent.*

	$\lambda_{\min}$	$\lambda_{\max}$	$\mathrm{cond}(\cdot)$
A	$1.00{\cdot}10^{-8}$	$1.27{\cdot}10^{-1}$	$7.86{\cdot}10^{8}$
Jacobi	$1.47{\cdot}10^{-4}$	$7.16{\cdot}10^{5}$	$2.06{\cdot}10^{10}$
GS	$1.00{\cdot}10^{0}$	$1.07{\cdot}10^{5}$	$9.26{\cdot}10^{6}$
Schur	$1.00{\cdot}10^{-8}$	$1.07{\cdot}10^{-3}$	$9.26{\cdot}10^{6}$

Table 6.1: *Largest and smallest eigenvalues of regularized and preconditioned system. Regularization parameter $\alpha = 10^{-8}$ and 8 dimensional subspace created by* regutm*. The problem is* heat *with $n = 64$.*

	$\lambda_{\min}$	$\lambda_{\max}$	$\mathrm{cond}(\cdot)$
A	$1.00{\cdot}10^{-8}$	$1.27{\cdot}10^{-1}$	$7.86{\cdot}10^{8}$
Jacobi	$1.39{\cdot}10^{-3}$	$1.38{\cdot}10^{5}$	$1.00{\cdot}10^{8}$
GS	$1.00{\cdot}10^{0}$	$2.29{\cdot}10^{2}$	$4.36{\cdot}10^{3}$
Schur	$1.00{\cdot}10^{-8}$	$2.29{\cdot}10^{-6}$	$4.36{\cdot}10^{3}$

Table 6.2: *Largest and smallest eigenvalues of regularized and preconditioned system. Regularization parameter $\alpha = 10^{-8}$ and 32 dimensional subspace created by* regutm*. The problem is* heat *with $n = 64$.*

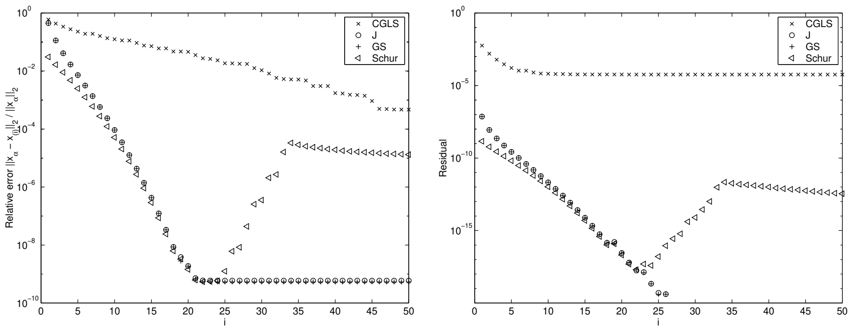
6.4 Convergence

We now turn to the actual convergence of our methods. We expect GS and Schur CG to perform equally well and Jacobi to perform worse than the unpreconditioned system when using a non-optimal subspace.

Figure 6.8 shows the convergence for Jacobi, GS and Schur CG with the optimal 32 dimensional SVD basis compared to standard CGLS. This optimal choice of subspace gives the methods best possible conditions. We see that Jacobi and GS performs equally as we had expected, cf. Section 4.2.3. Schur CG has a better initial error than the two preconditioners, but GS and Jacobi quickly catch up. However, Schur CG diverges when it has achieved its best possible precision after 24 iterations indicating that a stopping criteria is strongly needed. This fact contradicts theory and indicates that something is wrong with the theory or the implementation[2]. The residual plot shows how this criteria easely could be implemented by monitoring the residual. Figure 6.9 shows the results of a similar experiment with the test problem heat. Due to a higher condition number the convergence is slower for all methods, but the the precondtioned methods

[2]This problem is discussed in the Appendix A.1

still perform better. Jacobi seems to be a bit slower than GS and Schur CG.



(a) Relative error. **(b)** Residual.

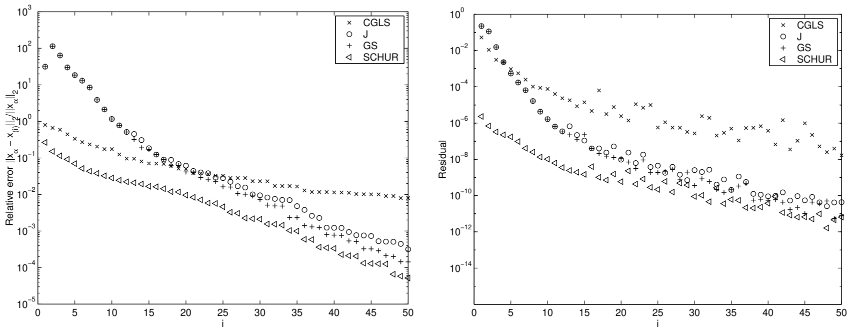
Figure 6.8: *Convergence for problem* deriv2 *with Jacobi, GS, Schur CG and CGLS on regularized system with $\alpha = 10^{-8}$. The subspace is a 32-dimensional SVD basis.*

We have in Figures 6.8 and 6.9 used the optimal subspace obtained from an SVD. If we now switch to a subspace obtained from regutm, we get Figures 6.10 and 6.11. Schur CG and GS show approximately the same convergence speed as with the optimal subspace and the initial error levels are also nearly the same. Jacobi on the other hand starts out with a much larger initial error and it takes many iterations before the final result is achieved.

6.5 The Regularization Parameter α

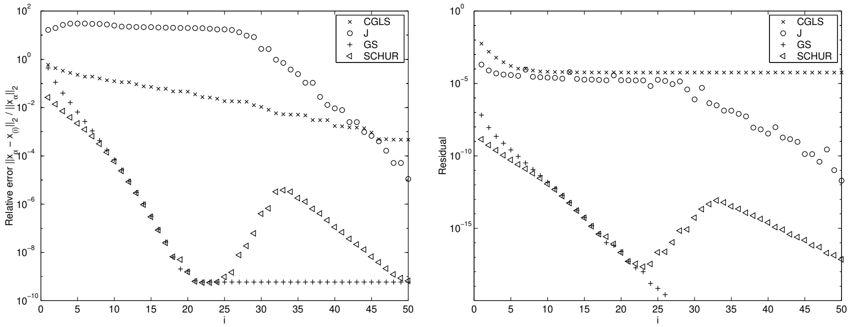
The choice of the regularization parameter α is not easy and will not be discussed in this thesis with this section as an exception. Here we take a look at the consequences of choosing a non-optimal α with respect to the *true* solution. Hence we will in this section experience semiconvergence, cf. Section 4.1.

Figure 6.12(b) shows the convergence of the two-grid methods with an optimal α along with the convergence of the CGLS on the *unregularized* problem. The optimal regularization parameter α is estimated by calculating



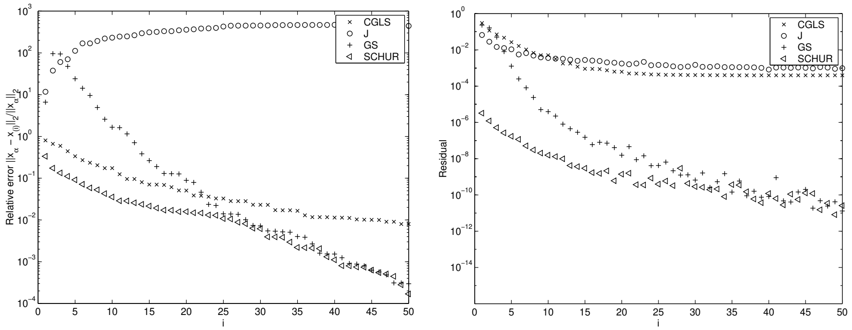
(a) Relative error. **(b)** Residual.

Figure 6.9: Convergence for problem heat *with Jacobi, GS, Schur CG and CGLS on regularized system with $\alpha = 10^{-8}$. The subspace is a 32-dimensional SVD basis.*



(a) Relative error **(b)** Residual

Figure 6.10: Convergence of Jacobi, GS, Schur CG and CGLS for regularized system. Subspace extracted from the regutm *function (32 dimensional). $\alpha = 10^{-10}$.*

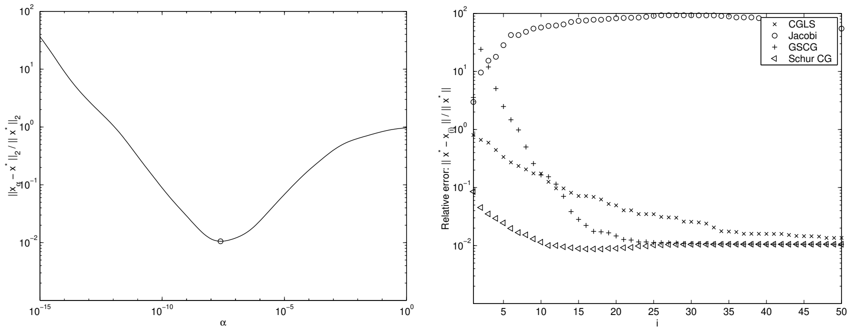


(a) Relative error **(b)** Residual

Figure 6.11: Convergence of Jacobi, GS, Schur CG and CGLS for regularized system. Subspace extracted from regutm *(32 dimensional). $\alpha = 10^{-10}$.*

200 Tikhonov regularized solutions and picking the one with the smallest error, see Figure 6.12(a). Other approaches include the L-curve criteria and generalized cross-validation [16, Chapter 7]. The unregularized CGLS method does not obtain a result better than Schur CG or GS within 64 iterations. Schur CG achieves a better result than the other methods. Furthermore we observe that Schur CG has a much lower initial error and that GS converges faster than CGLS. Jacobi looses all signs of convergence within the plotted iterations and seems very sensitive to noise — after 400 iterations Jacobi finally reaches the same level as GS and Schur CG.

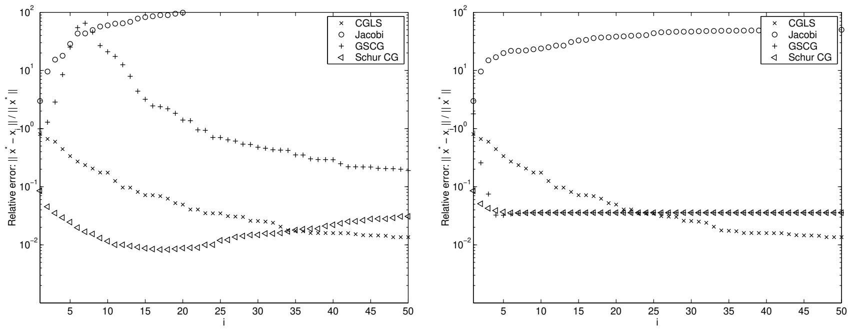
We now consider the case where α is chosen smaller than the optimal. This choice seems wise as we know CG has an regularizating effect which now is used together with the regularization we enforce throug α. Figure 6.13(a) shows the convergence with an α smaller than the optimal. Schur CG has the expected semi-convergence before the method finds the regularized solution. GS on the other hand does not show semiconvergence before finding the regularized solution and Jacobi does (again) not converge within the observed number of iterations. Figure 6.13(b) shows the complementary case with an α chosen to large. In this case the regularized methods except Jacobi converge fast but the solution is overregularized while CGLS is able to obtain a better solution. In all cases Jacobi takes a detour of nearly 400 iterations before it finds the regularized solution.



(a) Error for Tikhonov solutions with different α. Optimal error marked with circle.

(b) Convergence with optimal α.

Figure 6.12: *Convergence of* heat *with respect to the* true *solution. The regulatization parameter $\alpha \approx 2.3 \cdot 10^{-8}$ and the subspace is a 16-dimensional and created by* regutm*. CGLS works on the unregularized problem.*



(a) Convergence with $\alpha = \alpha_{\text{opt}}/1000$.

(b) Convergence with $\alpha = 1000\alpha_{\text{opt}}$.

Figure 6.13: *Convergence of* heat*. The 16-dimensional subspace is created by* regutm*. CGLS works on unregularized system.*

6.6 Filter Factors

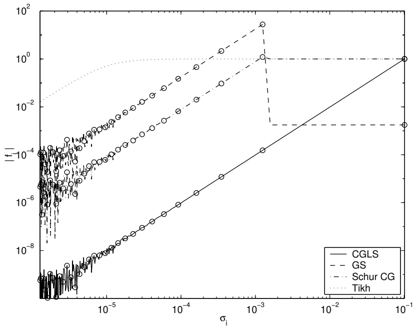
The filter factors (Section 2.2.1) tell which "Fourier components" that are found in terms of the SVD. The plots include the Tikhonov filter factors that both CGLS (on the regularized system) and Schur CG should converge towards. In Figure 6.14 we see the convergence using a SVD basis. Schur CG has after the first iteration found the components belonging to the first 8 singular values, while GS has the same components but on a constant lower level. At all iterations Schur CG performs better than GS, which in turn seems better than CGLS, in the sense that more SVD components are included in the case of Schur CG.

If we turn to a suboptimal subspace from regutm and find the filter factors we get Figure 6.15. The overall picture is the same. Both experiments show that GS seems to capture the Tikhonov filter factors of the small singular values at a very early point. Mayby this is the reason that semiconvergence was not observed in the previous section — recall that GS had a cluster of eigenvalues 1 with the smallest eigenvalues of the Schur complement (divided by α) being just a bit larger. When the CG iterations progress the minimization property prefers to approximate the clustered eigenvalues at 1 and therefore are the small Schur eigenvalues also approximated well. Another observation is that GS have filter factors (much) greater than 1.

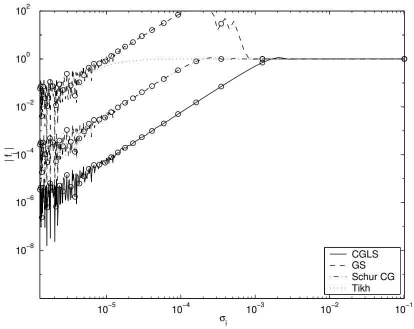
6.7 Subspace Choices

In this section we compare the subspace types discussed in Chapter 5. We have previously seen results for the SVD and the regutm basis. This section shows experiments with the remaining subspace types mentioned in Chapter 5. The wavelet basis is given special attention due to the choice of the genus D.

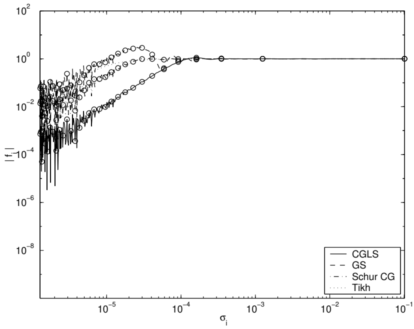
The SVD basis is used as reference, because of its optimality, together with the standard CGLS. We only do experiments with Schur CG because we have in the previous sections seen that it performs better than GS, albeit only with a small margin.



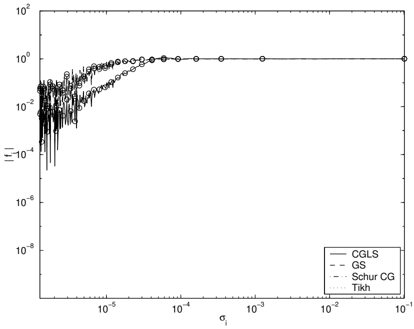
(a) Iteration 1.



(b) Iteration 8.

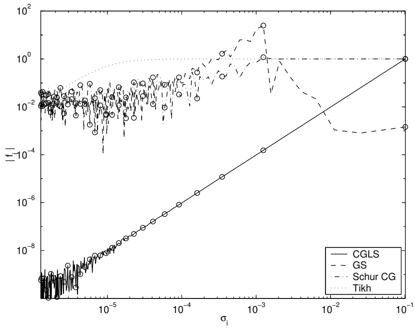


(c) Iteration 32.

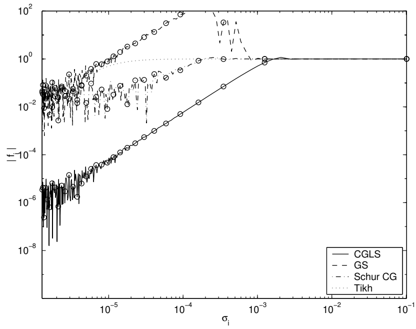


(d) Iteration 64.

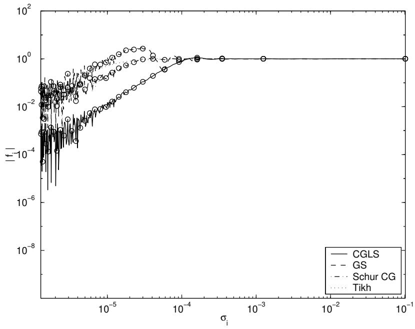
Figure 6.14: *Filter factors for* deriv2*. Subspace from SVD with dimension 8. $\alpha = 10^{-10}$ which means filtering for the singular values $\sigma_i \leq 10^{-5}$ indicated by the Tikhonov filter factors. Noise with standard deviation $10^{-8}\|\mathbf{b}\|_2$ is added to* **b***. Every 8th singular value is marked by a circle.*



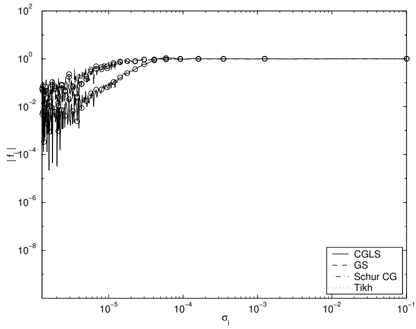
(a) Iteration 1.



(b) Iteration 8.



(c) Iteration 32.



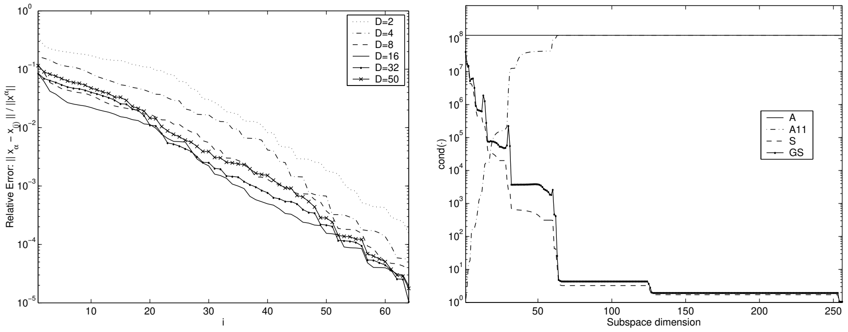
(d) Iteration 64.

Figure 6.15: *Filter factors for* deriv2*. Subspace from* regutm *with dimension 8. $\alpha = 10^{-10}$ which means filtering for the singular values $\sigma_i \leq 10^{-5}$ indicated by the Tikhonov filter factors. Noise with standard deviation $10^{-8}\|\mathbf{b}\|_2$ is added to* **b***. Every 8th singular value is marked by a circle.*

Wavelet Bases

A wavelet can have many forms according to its family and other parameters. We will only use the "standard" Daubechies wavelets of various genus. Figure 5.1 shows that wavelets of genus 2 and 4 were less smoth than those of genus 8 and 16. Hence we can expect a high genus to perform better than a small.

Figure 6.16(a) shows the convergence for 6 different genuses. Genus $D = 8$ is almost as good as $D = 16, 32, 50$ and the transformations are cheaper when using a fast wavelet transform. Hence we will use $D = 8$ in the remaining of this thesis. A higher genus is costly in terms of computation and it seems that the profit is low.



(a) Convergence using different genus with subspace dimension 16.

(b) Condition number as function of subspace dimension ($D = 8$).

Figure 6.16: *Convergence using different types of wavelets and condition number as function of subspace dimension.*

In Section 5.1.3 it was argued that only subspace dimensions that are powers of two should be choosen. Figure 6.16(b) shows the condition numbers as a function of subspace dimension (similar to Figure 6.6) and jumps are visible at powers of two.

Notes on other Subspace Choices

The Lanczos vectors were created ahead of the iterations using Algorithm 9.2.1 from [7]. This gives the algorithm a little advantage compared to

the "boot strap" idea, mentioned in Section 5.1.4, because the algorithm has the Lanczos vectors ahead of time. On the other hand these Lanczos vectors are different than those from a bootstrap algorithm because they are constructed from the unchanged and unpreconditioned system.

Condition Numbers

Table 6.3 gathers the condition numbers of Jacobi, GS, Schur and $\mathbf{A}_{11}$ for a fixed subspace dimension but using different subspace types. The condition number of the regularized system is included as a reference. We observe that the Lanczos subspace perform well even if noise has been added to the right-hand side, that the Lanczos vectors are created from.

	Jacobi	GS	Schur	$\mathbf{A}_{11}$
SVD	$3.14{\cdot}10^1$	$3.14{\cdot}10^1$	$3.14{\cdot}10^1$	$2.97{\cdot}10^3$
regutm	$2.64{\cdot}10^7$	$2.21{\cdot}10^2$	$1.00{\cdot}10^2$	$1.37{\cdot}10^4$
Lanczos	$1.36{\cdot}10^2$	$6.84{\cdot}10^1$	$4.83{\cdot}10^1$	$9.05{\cdot}10^3$
Lanczos w. noise	$2.01{\cdot}10^2$	$6.88{\cdot}10^1$	$4.90{\cdot}10^1$	$8.93{\cdot}10^3$
Wavelet ($D = 8$)	$1.11{\cdot}10^8$	$6.66{\cdot}10^1$	$4.18{\cdot}10^1$	$1.56{\cdot}10^4$
Sine/Cosine	$9.11{\cdot}10^7$	$6.15{\cdot}10^1$	$3.63{\cdot}10^1$	$7.53{\cdot}10^3$
Chebyschev	$8.76{\cdot}10^3$	$8.99{\cdot}10^2$	$1.08{\cdot}10^2$	$1.26{\cdot}10^5$
$\mathbf{K}^T\mathbf{K} + \alpha\mathbf{L}^T\mathbf{L}$	$1.26 \cdot 10^5$			

Table 6.3: *Condition numbers with different subspace types (*heat*). Subspace dimension is 16 and $\alpha = 10^{-6}$. The subspace "Lanczos with noise" is created from the right-hand side added normally distributed noise with standard deviation 10^{-4}.*

The regutm and Chebyshev bases show higher condition numbers than the rest. The Lanczos and Chebyshev bases show two, somewhat surprisingly, small condition numbers when used with Jacobi.

Convergence

Figure 6.17 shows the convergence for heat using 16-dimensional subspaces of all discussed types. We see that the Lanczos subspaces created with and without a noisy right-hand side perform just as good as the SVD basis. The Chebyshev and Wavelet bases both start out with a higher error and the convergence rate seems a bit smaller.

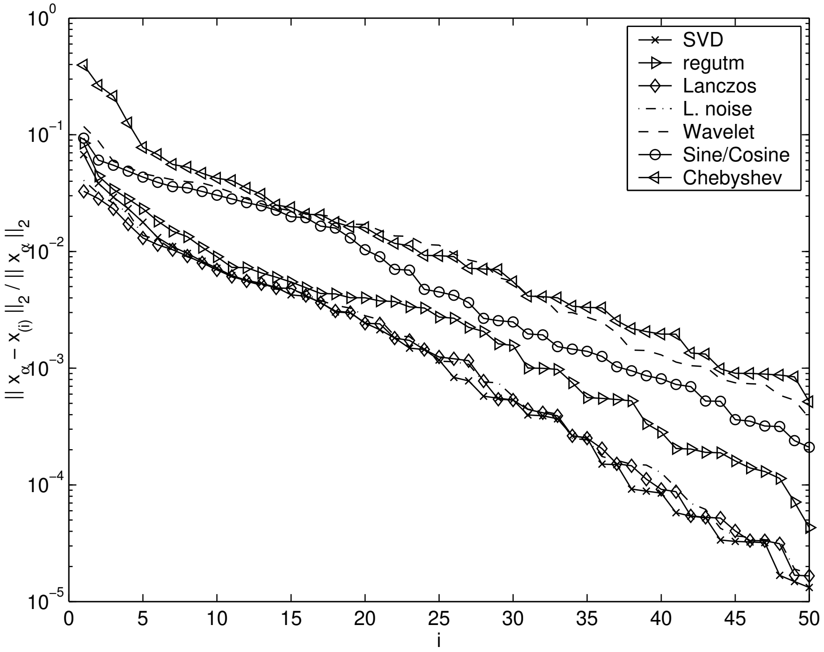
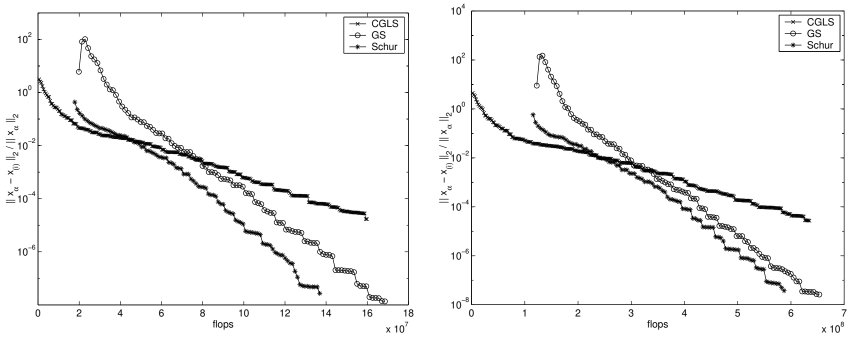


Figure 6.17: *Convergence for* heat *using different subspace types. Problem size is $n = 256$ with $\alpha = 10^{-6}$ and subspace dimensions 16.*

6.8 Counting Flops

Until now the convergence has been measured per iteration. This approach handicaps the unpreconditioned method because it is faster to do an iteration with CGLS than with, e.g., Schur CG. To see if anything is actually gained the convergence should be measured with respect to computation time or flops (floating point operations). We have used flops because flops are invariant with respect to the machine used.



(a) Problem size $n = 256$ **(b)** Problem size $n = 512$

Figure 6.18: *Convergence as a function of flops using* heat *and subspace* regutm*. Both cases uses $\alpha = 10^{-8}$ and $\mathbf{L} = \mathbf{I}_n$. Note that CGLS operates with $\mathbf{K} = [\mathbf{K}^T \sqrt{\alpha}\mathbf{I}_n]^T \in \mathbb{R}^{2n\times n}$.*

Figure 6.18 shows a clear victory to Schur CG. GS is handicapped by using 3 applications of the system matrix $\mathbf{K}$ per iteration, while Schur CG only uses 2. Note that this comparison is not entirely fair to CGLS as optimizations possible because of the special system matrix $[\mathbf{K}^T \sqrt{\alpha}\mathbf{I}_n]^T$ has not been implemented. However, the conclusion with respect to GS and Schur CG is still valid.

6.9 Variations of the Algorithms

Three specializations of the algorithms have been developed:

- A 2-D Schur CG method. The algorithm uses the Kronecker product approach explained in Section 5.2.

- A Schur CG method using wavelet transforms. The generation of $\mathbf{A}_{11}$ and the transformation to and from the coarse subspace $\mathcal{V}_k$ is implemented using fast wavelet transforms.
- A Jacobi and a Gauss-Seidel two-grid preconditioned method applied to the *unregularized* problem.

Matlab code and documentation is available in the appendix. In the following we investigate the results of the changes.

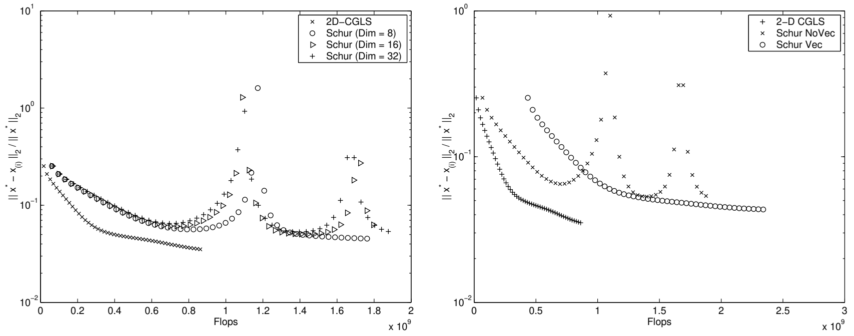
6.9.1 The Kronecker Variation

The test problem blur is created from a Kronecker product and thus forms a perfect test problem. The test problem had the size $\mathbf{X} \in \mathbb{R}^{128\times 128}$ which in a non-Kronecker situation would require a kernel $\mathbf{K} \in \mathbb{R}^{16384\times 16384}$. Figure 6.19(a) shows the convergence of a 2-D CGLS[3] compared to those of Schur CG using an approximation to $\mathbf{A}_{11}$. In this particular example 2-D CGLS is preferable because it converges faster. Schur CG shows some strange jumps, which must be caused by the approximation of $\mathbf{A}_{11}$, cf. Figure 6.19(b), that shows a comparison of Schur CG with and without the approximation of $\mathbf{A}_{11}$. The approximation speeds up the calculations but at the expense of stability. Table 6.4 shows the costs of each of the methods. Included is an estimation of the costs of not using the standard CGLS and Schur CG methods and the estimations clearly show that one should be patient if such an experiment is attempted. We see that the speedup caused by the approximated $\mathbf{A}_{11}$ is mostly limited to the initialization phase.

Method	Flop/Iteration	Initial Costs
2-D CGLS	$1.70 \cdot 10^7$	$8.42 \cdot 10^6$
Kronecker Schur CG	$3.89 \cdot 10^7$	$3.96 \cdot 10^8$
Kronecker Schur CG (novec)	$3.69 \cdot 10^7$	$3.08 \cdot 10^7$
CGLS (estimate)	$> 1 \cdot 10^9$	$> 5 \cdot 10^8$
Schur CG (estimate)	$> 2 \cdot 10^9$	$> 6 \cdot 10^{11}$

Table 6.4: *Iteration and startup costs using the Kronecker variation compared to other more general algorithms. The subspace dimension is* 32^2.

[3] The standard CGLS algorithm modified to take advantage of the Kronecker structure



(a) CGLS vs. Schur CG (novec).

(b) True Schur CG vs. the vector loose version. Subspace dimension is 32.

Figure 6.19: *Convergence using the Kronecker variation. The test problem is* blur *with* $n = 128$, $\sigma = 0.8$ *and the regularization parameters are* $\mathbf{L} = \mathbf{I}_{n^2}$ *(actually a Kronecker product) and* $\alpha = 10^{-3}$.

6.9.2 Schur CG with Wavelets

The Matlab function in Appendix A.5.1 implements Schur CG where the operation of $\mathbf{\Phi}$ is performed by fast wavelet transforms.

Figure 6.20 shows the benefit using the wavelet transform on a big problem, which in this case is heat with $\mathbf{K} \in \mathbb{R}^{1024\times 1024}$. The initial computations are cheaper than using standard matrix operations and each iteration is a bit cheaper. An increase in subspace dimension increases the size of $\mathbf{A}_{11}$ but the operations with $\mathbf{\Phi}$ are not more costly. The profit is definitely noticeable and the wavelet approach seems very promising.

6.9.3 Non-Regularized Problems

The last modification considered is to apply the preconditioners to the unregularized problem. In this case we only use CG's semiconvergence property as regularization tool.

Figure 6.21 shows the (semi)-convergence of the Jacobi and Gauss-Seidel two-grid preconditioned systems compared to CGLS. If only the number of iterations is considered we see a slightly better performance of the GS

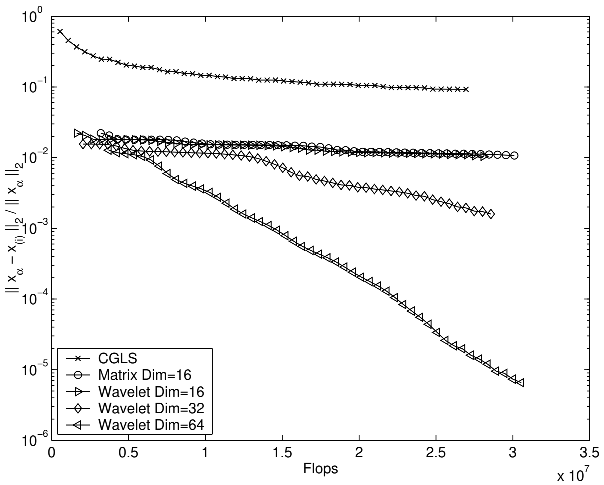


Figure 6.20: *Convergence vs. flops with wavelets. The problem size is $n = 1024$ and 50 iterations are performed. CGLS is working on the unregularized problem.*

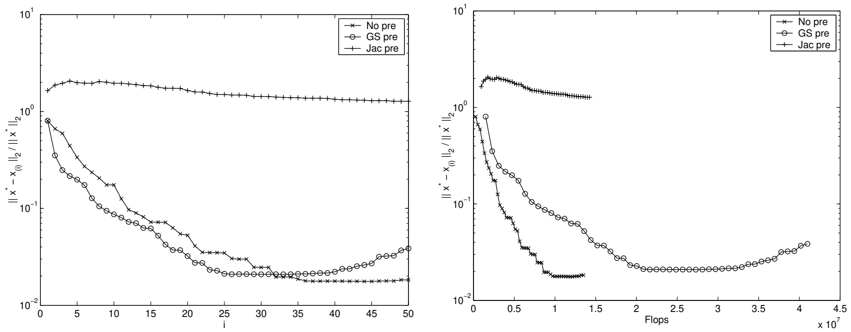
preconditioner, while Jacobi is far behind. However, if the amount of work per iteration is considered CGLS turns out to be the best choice.

6.10 The Large Problems

We end the numerical experiments with the two larger problems vesuvio and geomig. The hope is that the good properties from the small problems used above carries over to the large problems.

6.10.1 Geophysical Migration

Many parameters for this test run are inspired from [30] because our knowledge of geophysics is limited.



(a) Convergence vs. iteration. **(b)** Convergence vs. flops.

Figure 6.21: *Convergence using preconditioners on the unregularized* heat *problem. Normally distributed noise with deviation 10^{-5} added such that semiconvergence is noticeable within the 50 iterations made. The regularization parameter $\alpha = 10^{-3}$ and the subspace splitting is a 8 dimensional* regutm *subspace.*

Beside the usual $\mathbf{L} = \mathbf{I}_n$ we used the regularization matrix

$$\mathbf{L} = \begin{bmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & 2 & -1 & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & -1 \\ 0 & \cdots & 0 & -1 & 2 \end{bmatrix} \otimes \mathbf{I}_{95} \tag{6.1}$$

This choice is based on the knowledge that most seismic profiles have horizontal boundaries. This holds true for our test example with the exception of the slope at the bottom (Figure 6.4).

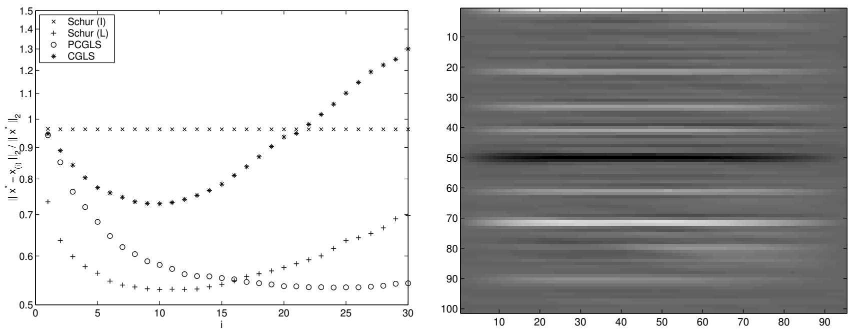
The subspace used is composed by a Kronecker product having 10 in the "horizontal kernel" (the left factor in the Kronecker product) and 10 in the "vertical kernel", that is

$$\mathbf{\Phi} = \mathbf{\Phi}_1 \otimes \mathbf{\Phi}_2, \quad \mathbf{\Phi}_1 \in \mathbb{R}^{95\times 10} \text{ and } \mathbf{\Phi}_2 \in \mathbb{R}^{101\times 10},$$

which yields $\mathbf{\Phi} \in \mathbb{R}^{9595\times 100}$. Both $\mathbf{\Phi}_1$ and $\mathbf{\Phi}_2$ are created using regutm.

Figure 6.22(a) shows the error with respect to the true solution and hence we observe semiconvergence. We see that Schur CG with (6.1) achieves the best solution after 10 iterations while the "preconditioned" CGLS (PCGLS) have its best solution after 23 iterations. On the other hand we note that CGLS performs better than Schur CG with the identity as regularization matrix. Table 6.5 show the flop-count per iteration and we see that Schur CG is twice as expensive per iteration and in that perspective PCGLS and Schur CG performs almost equal. However, if we also consider the initial costs PCGLS wins with a large margin. The initial costs, most notable the creation of $\mathbf{A}_{11}$, are to large for Schur CG to compete in this example. Finally we see that the cost of using the more complicated $\mathbf{L}$ is almost negligible.

Figure 6.22(b) shows the best solution from Schur CG with (6.1). The horizontal layers are found while the slope at the bottom is missing. This was expected because of the regularization matrix.



(a) Convergence history. **(b)** Best solution.

Figure 6.22: *Convergence and best solution for* geomig. *The best solution is obtained with Schur CG with (6.1) and is reached after 10 iterations. PCGLS reaches its best solution after 23 iterations, but it is not as good as that of Schur CG. The parameter $\alpha = 1$ is selected by trial and error and noise with standard deviation 10^{-4} is added to the right-hand side.*

6.10.2 Inversion of Geomagnetic Data

An SVD decomposition of the test problem vesuvio reveals that the first right singular vectors only include information on the top most layer, i.e.,

Algorithm	Flops/Iteration	Initial Costs
CGLS	$3.28 \cdot 10^7$	$1.56 \cdot 10^7$
PCGLS	$3.26 \cdot 10^7$	$3.16 \cdot 10^7$
Schur CG $\mathbf{I}_n$	$6.67 \cdot 10^7$	$4.01 \cdot 10^9$
Schur CG $\mathbf{L}$	$6.63 \cdot 10^7$	$4.21 \cdot 10^9$

Table 6.5: *Iteration and initial costs for* geomig. *The initial work of CGLS and PCGLS amounts to a half iteration and one iteration respectively.*

the layer just beneath the surface. As a consequence CGLS have difficulties reconstructing information deeper below the surface. The solution is to use another regularization matrix $\mathbf{L}$ that modifies the problem in a way that information information on deeper components are present in the the first generalized right singular values (the *last* columns of GSVD's $\mathbf{X}$). Experiments show that the choice

$$\mathbf{L} = \mathbf{L}_1 \otimes (\mathbf{L}_1 \otimes \mathbf{L}_1),$$

where

$$\mathbf{L}_1 = \begin{bmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & 2 & -1 & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & -1 \\ 0 & \cdots & 0 & -1 & 2 \end{bmatrix}$$

has the desired effect.

Figure 6.23 shows the convergence of Schur CG and PCGLS with the special regularization matrix. Furthermore we did the experiment with standard CGLS and Schur CG using the identity. We see Schur CG with $\mathbf{L}$ achieves a better result than the others. PCGLS shows the second best result. CGLS shows very slow convergence just as Schur CG using the identity.

Table 6.6 lists the cost of an iteration and the initial costs of each method. The high initial costs handicaps Schur CG, but the cost per iteration is almost 1/4 of that of PCGLS. Thus if many iterations are needed Schur CG with the special regularization matrix Schur CG is preferable. We observe from tables 6.5 and 6.6 that Schur CG's initial costs are relatively larger than the cost of an iteration when the system matrix is sparse.

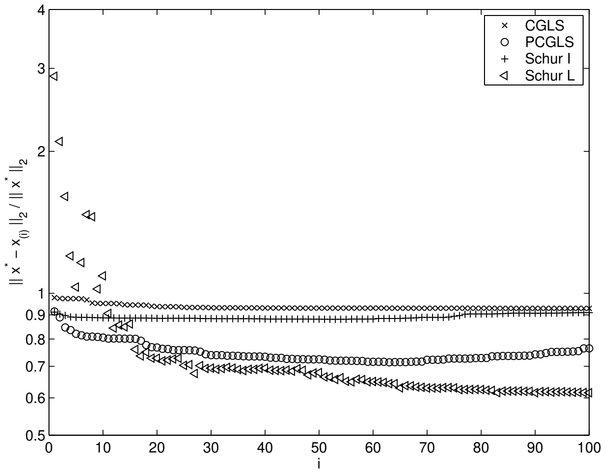


Figure 6.23: *Convergence plot for* vesuvio*.*

Algorithm	Flops/Iteration	Initial Costs
CGLS	$4.03 \cdot 10^6$	$2.00 \cdot 10^6$
PCGLS	$3.50 \cdot 10^7$	$3.50 \cdot 10^7$
Schur CG $\mathbf{I}_n$	$8.29 \cdot 10^6$	$1.43 \cdot 10^8$
Schur CG $\mathbf{L}$	$9.23 \cdot 10^6$	$2.63 \cdot 10^8$

Table 6.6: *Iteration and initial costs for* vesuvio*. The initial work of CGLS and PCGLS amounts to a half iteration and one iteration respectively.*

Chapter 7

Conclusion

This final chapter forms a summary on the performed work and the lessons learned.

The conjugate gradient method has been explained and its convergence properties have been covered. We introduced the convergence in a general setting and we looked at the special features with respect to ill-posed problems and the need for preconditioning has been motivated.

The two-grid methods proposed by Hanke and Vogel have been explained in detail. The algorithms for the restricted class of regularization matrices have been generalized to support all semidefinite $\mathbf{L}^T\mathbf{L}$. We have estimated the condition numbers of the treated methods and hereby we found, that the Jacobi preconditioned system is likely to converge slower than even the unpreconditioned system.

Through knowledge of the usual behaviour of the right singular vectors we have proposed a number of subspace splittings suitable for use with the algorithms.

We have implemented the Schur complement, the Jacobi-like two-grid preconditioned and the Gauss-Seidel-like two-grid preconditioned conjugate gradient methods in two versions; one for a full rank regularization matrix $\mathbf{L}$ and one for a regularization matrix with a nontrivial null-space. Furthermore we have implemented two variations of Schur complement conjugate gradients that utilizes Kronecker products and wavelet transforms respectively. The preconditioned methods have the possibility to

work with the unregularized methods and hereby rely on semiconvergence to perform the regularization.

The algorithms have undergone numerical tests in order to show strengths and weaknesses. The following is a list of observations made from the experiments:

- The Jacobi-like two-grid preconditioner is very sensitive to the selected subspace and noise in the right-hand side. In some cases the convergence is much slower than the standard CGLS method.
- The Gauss-Seidel-like two-grid preconditioner and the Schur complement conjugate gradient converge equally fast when measuring iterations. However, an iteration with Schur is approximately $2/3$ the work of that of Gauss-Seidel.
- The choice of the regularization parameter α should rather be to low, as Schur conjugate gradients also shows semiconvergence.
- The optimal subspace choice is based on an SVD. However, the subbases Lanczos and `regutm` do not enforce a great penalty. The sine/-cosine, wavelets and Chebyshev bases seem a bit weaker.
- The use of fast wavelet transforms enable the use of larger subspaces. A larger subspace is desirable because of faster convergence. However, the fast wavelet transform requires the problem size to be a power of 2.
- The preconditioners does not work well with the unregularized problem.
- The "Kronecker" variation of the Schur complement algorithm did not prove usefull for the considered test problem `blur`.

The general impression is that much care must be taken to select the subspace splitting and that the standard CGLS method in many cases is a better choice. However, we did see a very good results in particular with the special wavelet implementation.

Future Work

We have in this thesis covered many facets of the algorithms but just as many (at least) are still uncovered. After having worked with the algorithms we conclude this thesis with a few ideas that seem interesting to investigate:

- An implementation using a fast Fourier transform. The fast Fourier transform does not require the problem size to be a power of two — a problem with the fast wavelet transform.
- The implementations of this thesis does not have stopping criteria besides a iteration limitation. A reliable stopping criteria is needed.
- A parallelization, if possible, of the algorithms is necessary in order to exploit the current super-computers.
- A "bootstrap" preconditioner that is iteratively extended from the Lanczos vectors as they become available in each iteration.
- We have not compared our preconditioners with others and it is interesting to see whether other preconditioners can do better and for which problems.

Appendix A

Matlab Code

Any code is useless without proper testing and documentation on its use. The next section describes how it has been tested that the actual implementations are correct with respect to the derivations from Chapter 4. Then two sections show a "manual-page" along with the actual Matlab code. The algortihms are divided into two sections according to the size of the null space of the regularization matrix $\mathbf{L}$. Even though the synopsis are almost the same for an empty null space and a non-empty null space and the manual pages are very alike they are repeated for completeness.

Finally we list two specializations of Schur complement CG, i.e., a implementation with the subspace operations done by fast wavelet transforms and a implementation using Kronecker products as kernel, subspace etc.

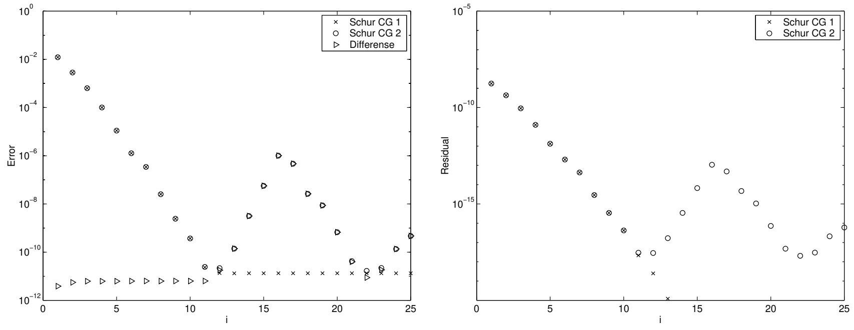
A.1 Test Procedures

The two preconditioners have been tested using the same procedure, while Schur CG has been tested in its own way because of its different nature.

A.1.1 Schur Complement CG

The test for the Schur complement CG method was to create the actual Schur complement $\mathbf{S}$ and operate entirely in the "Schur space" and not

transform all vectors back into the normal basis. Thereafter we compared the results $\mathbf{x}_{(k)}$, residuals $\mathbf{r}_{(k)}$ etc. computed by the actual programs by transforming the "Schur space" vectors back.



(a) Error and difference of Schur algorithms.

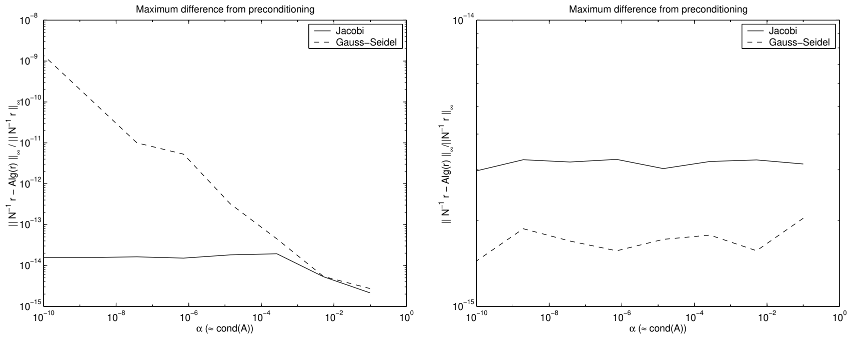
(b) Residuals

Figure A.1: *Verification of Schur complement CG algorithm. Schur CG 1 denotes a Schur algorithm computed in the Schur complement space and afterwards transformed. Schur CG 2 is our method. "Difference" is the norm of the difference of the solution in each iteration. The test problem is* deriv2 *with* $n = 64$ *and* $\alpha = 10^{-7}$ *with a wavelet basis of dimension 16.*

Figure A.1(a) shows that our algorithms at a certain point diverges from the solution which the untransformed Schur CG finds. Figure A.1(b) show the residuals and gives the explanation of the strange behaviour. The two versions of Schur CG both have the same residual until the norm reaches the machine precision. At this point the untransformed continues to reduce its residual while our method "stumbles". The reason must be that we use the residual with, e.g. $\mathbf{M}^{-1}\mathbf{r}$ and that we can not expect the residual to be precise at this point. This means that the updates of $\mathbf{M}^{-1}\mathbf{r}$ are now out of synchronization and we can not expect convergence.

A.1.2 The Preconditioned Methods

We assume the actual CG part of the algorithms to be correct and thus we only need to test the preconditioner. We construct the block preconditioner $\mathbf{N}$ explicitly (both the Jacobi and the Gauss-Seidel like) and we compute $\mathbf{N}^{-1}\mathbf{r}$ for some random $\mathbf{r}$ using the Matlabs standard "backslash" operator. Then we take the same $\mathbf{r}$ and put it through the preconditioning part of the CG algorithm and compare the result to the simple version. The preconditioning part was extracted from the algorithms for this purpose. The result of the comparisons seems to be linked to the condition number of $\mathbf{K}^T\mathbf{K} + \alpha\mathbf{L}^T\mathbf{L}$. Figure A.2 shows the difference as a function of the condition number of the regularized system. The differences can be explained by the fact that the the preconditioning algorithms incorporates subtraction operations with the danger of cancellations. The nicer behaviour obtained with an SVD basis must be contributed the fewer complications arising from diagonal matrices.



(a) Wavelet basis **(b)** SVD basis

Figure A.2: *Comparison of algorithms with their matrix equivalents as a function of α and hence the condition number* $\mathrm{cond}(\mathbf{A})$.

A.2 A Short Implementation Note on $\mathrm{M}^\dagger$

The solution of systems with the pseudo-inverse of $\mathbf{L}^T\mathbf{L}$ is done as follows. A QR-factorization of $\mathbf{L} = \mathbf{QR}$ is calculated from which a factorization of $\mathbf{L}^T\mathbf{L} = \mathbf{R}^T\mathbf{R}$ is easely obtained. A "backslash" operation with $\mathbf{R} \in \mathbb{R}^{p\times n}$ and $\mathbf{e} \in \mathbb{R}^p$ where $p < n$ selects the solution (out of infinitely many) with a minimum of $n - p$ zeros. This coresponds to the solution $\mathbf{w} = \mathbf{R}^\dagger\mathbf{e} + \boldsymbol{\Phi}_0\mathbf{z}$, where $\mathbf{z} \in \mathbb{R}^{n-p}$ and the columns of $\boldsymbol{\Phi}_0$ span the null space of $\mathbf{L}$. The component in the null space is then removed by $\mathbf{R}^\dagger\mathbf{e} = \mathbf{w} - \mathbf{P}_0\mathbf{w}$.

The "backslash" with the transposed $\mathbf{R}^T$ calculates the least squares solution and has no need for correction. The calculation is done with the aid of a possible expensive QR factorizaion of $\mathbf{R}^T$. However, the (usual) sparsity of $\mathbf{L}$ and hence $\mathbf{R}$ has keept the costs down and the problem was not noticed until late in the project and has therefore not been solved properly.

A.3 Symmetric Positive Definite $\mathbf{L}^T\mathbf{L}$

A.3.1 jacobicg

Purpose Conjugate gradient method with a two-grid Jacobi-like regularized preconditioner.

Synopsis [X, rho, eta, flopc] = jacobicg(K,Vk,alpha,L,b,maxiter{{,reg},trace})

Description Performs maxiter iterations of conjugate gradients with a two-grid block diagonal preconditioner (Jacobi-like), either on the unregularized normal equation system

$$\mathbf{K}^T\mathbf{K}\mathbf{x} = \mathbf{b} \qquad \text{if reg} \neq 1$$

or on the regularized system (default)

$$(\mathbf{K}^T\mathbf{K} + \alpha\mathbf{L}^T\mathbf{L})\mathbf{x} = \mathbf{K}^T\mathbf{b} \qquad \text{if reg} = 1$$

The preconditioner is formed from the subspace Vk, the ill-posed matrix K, the regularization matrix L and the regularization parameter alpha.

If the regularization matrix is the identity use the empty matrix [] as argument to achieve computational savings.

The result of each iteration is returned as columns in X. The residual of each iteration is stored in rho and eta respectively. A flop-count after each iteration is returned in flopc

If parameter trace is used, an estimation of remaining calculation time is printed after each iteration — useful for large problems.

Code

```
function [X, rho, eta, flopc] = ...
    jacobicg(K,V,alpha,L,b,maxiter,reg,trace)
% JACOBICG  Conjugated gradients with two-level Jacobi
%           like preconditioner for ill-posed problems.
%
% [X,rho,eta,flopc] =
%        jacobicg(K,V,alpha,L,b,maxiter {{,reg},trace})
%
```



```
% Solves a ill-posed problem with or without tikhonov
% regularization, i.e.,
%
% (K'K + alpha*L'L)u = K'b         (1)
%    or
% K'K u = K'b                      (2)
%
% Input arguments:
%  K     : Kernel matrix
%  V     : Basis for coarse subspace
%  alpha : Regularization parameter used in preconditioner
%          and in (1).
%  L     : Seminorm matrix (SPD)
%          Use [] for identity
%  b     : Right hand side vector
%  iter  : Maximum number of iterations
%  reg   :  =1 Use CG on (1)
%          ~=1 Use CG on (2)
%          Optional, default: 1
%  trace : Any value yields output information during
%          iterations; usefull for large problems.
%          Optional, default: off
%
% Output arguments:
%  X     : Solutions stored as columns
%  rho   : Residual norms
%  eta   : Solution norms
%  flopc : Flopcount after eacher iteration
%
% References:
% [1] Michael Jacobsen: Two-Grid Iterative Methods for
%                       Ill-Posed Problems

% Last Revised: 2000.09.02

% Checking input arguments and setting defaults
identity = 0;
if nargin < 8
  trace = 0;
  if nargin < 7
    reg = 1;
  end
end
if isempty(L)
  identity = 1;
end
if nargout == 4
  flops(0);
end
```

```
% Initial work (factorizations etc.)
b = K'*b;
KV = K*V;
if identity
  A11 = KV'*KV + alpha*V'*V;
  clear KV;
else
  LV = L*V;
  A11 = KV'*KV+alpha*LV'*LV;
  clear KV LV;
end
% Cholesky of SPD A11
R = chol(A11);

if identity
  G = V'*V;
else
  G  = V'*L'*L*V;
  LR = qr(L);       % L usually banded
end
GR = chol(G);

% Allocate memory
u = zeros(size(K,2),1);
X = zeros(size(K,2),maxiter);
rho = zeros(maxiter,1);
r = b;  delta = 0;

for i=1:maxiter
  % Preconditioning step
  v = V'*r;                        % Transform
  e = R\(R'\v) - GR\(GR'\v) / alpha;
  if identity
    w = r;                         % M\r;
  else
    w = LR\(LR'\r);                % M\r;
  end
  z = V*e + w/alpha;
  % Preconditioning step end

  % Cg step
  deltaold = delta;
  delta    = r'*z;
  if i == 1
    p = z;
  else
    beta = delta / deltaold;
    p = z + beta*p;
```



```
  end
  d = K'*(K*p);
  % Work on regularized system or not
  if reg == 1
    if identity
      d = d + alpha*p;            % Non regularized
    else
      d = d + alpha*(L'*(L*p));   % Regularized
    end
  end

  % Update data
  alfa = delta / (p'*d);
  u = u + alfa*p;
  X(:,i) = u;
  r = r - alfa*d;

  % Compute norms, if required
  if nargout > 1, rho(i) = norm(r); end
  if nargout > 2, eta(i) = norm(u); end
  flopc(i) = flops;

  % Print trace information
  if trace
    disp(['Iteration ', num2str(i), ' in progress'])
    if i > 1
      disp(sprintf('Done in %3.1f minutes', ...
  toc/60*(maxiter - i)));
    end
    tic
  end
end
```

A.3.2 gscg

Purpose Conjugate gradient method with two-grid symmetric Gauss-Seidel-like regularized preconditioning.

Synopsis [X, rho, eta, flopc] = gscg(K,Vk,alpha,L,b,iter{{,reg},trace})

Description Performs maxiter iterations of conjugate gradients with a two-grid block symmetric Gauss-Seidel-like preconditioner, either on the unregularized normal equation system

$$\mathbf{K}^T\mathbf{K}\mathbf{x} = \mathbf{b} \qquad \text{if reg} \neq 1$$

or on the regularized system (default)

$$(\mathbf{K}^T\mathbf{K} + \alpha\mathbf{L}^{\mathbf{L}})\mathbf{x} = \mathbf{K}^T\mathbf{b} \qquad \text{if reg} = 1$$

The preconditioner is formed from the subspace Vk, the ill-posed matrix K, the regularization matrix L and the regularization parameter alpha.

If the regularization matrix is the identity use the empty matrix [] as argument to achieve computational savings.

The result of each iteration is returned in columns of X and the norm of the residuals and solutions is stored in rho and eta respectively. A flop-count after each iteration is stored in flopc.

If the optional parameter trace is stated an estimated remaining calculation time is printed after each iteration.

Code

```
function [X,rho,eta,flopc] = ...
    gscg(K,V,alpha,L,b,iter,reg,trace)
% GSCG  Conjugated gradients with two-level preconditioner
%       Gauss-Seidell like preconditioner for ill-posed
%       problems.
%
% [X, rho, eta, flopc] =
%                GSCG(K,Vk,alpha,L,b,maxiter,reg)
%
% Solves a ill-posed problem with or without tikhonov
% regularization, i.e
%
```



```
% (K'K + L'L)u = K'b                 (1)
%    or
% K'K u = K'b                        (2)
%
% using conjugated gradients with a two grid Gauss-Seidel
% like regularized precondtioner.
%
% Input arguments:
%  K     : Kernel matrix
%  V     : Basis for coarse subspace
%  alpha : Regularization parameter used in preconditioner
%          and in (1).
%  L     : Seminorm matrix (SPD)
%          Use [] for identity
%  b     : Right hand side vector
%  iter  : Maximum number of iterations
%  reg   :  =1 Use CG on (1)  (Default)
%          ~=1 Use CG on (2)
%  trace : Output estimated remaining time after each iteration
%          Default off
%
% Output arguments:
%  X     : A matrix with all iter solutions stored as columns
%  rho   : A vector containing residuals
%  flopsc: A flopcount after each iteration.
%
% References:
% [1] Michael Jacobsen: Two-Grid Iterative Methods
%                       for Ill-Posed Problems

% Last Revised: 2000.09.02

% Checking input arguments
identity = 0;
if nargin < 8
  trace = 0;
  if nargin < 7
    reg = 1;
  end
end
if isempty(L)
  identity = 1;
end
if nargout == 4
  flops(0);
end

% Initial work (factorizations etc.)
b = K'*b;
```

```
KV = K*V;
if identity
  A11 = KV'*KV + alpha*V'*V;
  clear KV;
else
  LV = L*V;
  A11 = KV'*KV+alpha*LV'*LV;
  clear KV LV;
end

R = chol(A11);
if identity
  G  = V'*V;
  GR = triu(qr(V));
else
  LV = L*V;
  G  = LV'*LV;
  GR = triu(qr(LV));
  LR = triu(qr(L));
  clear LV;
end


% Allocate memory
u = zeros(size(K,2),1);
X = zeros(size(K,2),iter);
rho = zeros(iter,1);
r = b;
delta = 0;

for i=1:iter
  % Preconditioning step
  v = V*(R\(R'\(V'*r)));
  e = r - K'*(K*v);
  if identity
    w = e;                              % M\e;
  else
    w = LR\(LR'\e);                     % M\e;
  end

  px = w - V*(GR\(GR'\(V'*e)));        % V*G\V'*e
  uq = px / alpha;
  y = r - K'*(K*uq);
  up = V*(R\(R'\(V'*y)));
  z = up + uq;

  % Cg step
  deltaold = delta;
  delta    = r'*z;
```



```
  if i == 1
    p = z;
  else
    beta = delta / deltaold;
    p = z + beta*p;
  end
  d = K'*(K*p);
  % Work on regularized system or not
  if reg == 1
    if identity
      d = d + alpha*p;
    else
      d = d + alpha*(L'*(L*p));
    end
  end

  alfa = delta / (p'*d);
  u = u + alfa*p;
  X(:,i) = u;
  r = r - alfa*d;

  % Update norms, if required
  if nargout > 1, rho(i)   = norm(r); end
  if nargout > 2, eta(i)   = norm(u); end
  flopc(i) = flops;

  % Print trace information
  if trace
    disp(['Iteration ', num2str(i), ' in progress'])
    if i > 1
      disp(sprintf('Done in %3.1f minutes', ...
  toc/60*(iter - i)));
    end
    tic
  end
end
```

A.3.3 schurcg

Purpose Schur complement conjugate gradient method with regularization for ill-posed problems.

Synopsis [X, rho, eta, flopc] = schurcg(K,V,alpha,L,b,maxiter{,trace})

Description Performs maxiter iterations of the Schur complement conjugate gradient method on the regularized system

$$(\mathbf{K}^T\mathbf{K} + \alpha\mathbf{L}^T\mathbf{L})\mathbf{x} = \mathbf{K}^T\mathbf{b} \tag{A.1}$$

The Schur complement CG operates on a subspace which is $\mathbf{L}^T\mathbf{L}$-orthogonal to the subspace defined by V.

If the regularization matrix is the identity use the empty matrix [] as argument to achieve computational savings.

The result of each iteration is returned in X and norm of the residuals and solutions are stored in rho and eta respectively.

If parameter trace is used an estimation of remaining running time is printed after each iteration.

Code

```
function [X, rho, eta, flopc] = ...
    schurcg(K,V,alpha,L,b,maxiter,trace)
% SCHURCG  Schur Complement Conjugated Gradients
%          Assumes L to be invertible.
%
%   [X, rho, eta, flopc] =
%            SCHURCG(K,V,alpha,L,b,maxiter {,trace})
%
% Solves a tikhonov regularized system of type
%       (K'K+alpha*L'L)u = K'*b
% using schur complement conjugated gradients -- see [1].
%
% Input arguments:
%  K     : Kernel matrix
%  V     : Basis for coarse subspace
%  alpha : Regularization parameter
%  L     : Seminorm matrix (SPD)
%          Use [] for identity
%  b     : Right hand side vector
%  iter  : Maximum number of iterations
%  trace : (Optional) Print information during calculations
```



```
%
% Output argument:
%  X     : A matrix with all iter solutions stored as columns
%  rho   : Norm of residuals
%  eta   : Solution norms
%  flopc : Flopcount after each iteration
%
% References:
% [1] Michael Jacobsen: Two-Grid Iterative Methods
%                       for Ill-Posed problems

% Last Revised: 2000.08.24

if nargin < 7
  trace = 0;
end
if nargout == 4
  flops(0);
end

% Cholesky factorization of K'K and L'L
if ~isempty(L)
  tt   = K*V;  tt2  = L*V;
  A11  = tt'*tt+alpha*tt2'*tt2;
  RL   = qr(L);
  clear tt tt2;
else
  tt = K*V;
  A11  = tt'*tt+alpha*speye(size(V,2));
end

% A11 is SPD (at least theoreticaly)
% Cholesky fast and stable
RA11 = chol(A11);
if trace
  disp('A11 done')
end
% We are working with normal equations
b = K'*b;

% Initialization
u = V*(RA11\(RA11'\(V'*b)));
if ~isempty(L)
  r = b - K'*(K*u) - alpha*(L'*(L*u));
  y = RL\(RL'\r);
else
  r = b - K'*(K*u) - alpha*u;
  y = r;
end
```

```
d = r;
z = y;
delta0 = y'*r;
X = [];

for i=1:maxiter
  if ~isempty(L)
    tt = K'*(K*z) + alpha*(L'*(L*z));   % A*z
  else
    tt = K'*(K*z) + alpha*z;            % A*z
  end
  v  = z - V*(RA11\(RA11'\(V'*tt)));
  if ~isempty(L)
    w  = K'*(K*v) + alpha*(L'*(L*v));   % A*v
    g  = RL\(RL'\w);
  else
    w  = K'*(K*v) + alpha*v;            % A*v
    g  = w;
  end
  tau = delta0 / (d'*g);
  u = u + tau*v;                        % Update solution
  X(:,i) = u;                           % Store solution
  r = r - tau*w;                        % Update residual
  y = y - tau*g;
  delta1= r'*y;
  beta = delta1 / delta0;
  delta0 = delta1;
  z = y + beta*z;                       % Update search dir
  d = r + beta*d;

  % Update norms, if required
  if nargout > 1, rho(i) = norm(r); end
  if nargout > 2, eta(i) = norm(u); end
  flopc(i) = flops;

  % Print trace information
  if trace
    disp(['Iteration ', num2str(i), ' in progress'])
    if i > 1
      disp(sprintf('Done in %3.1f minutes', ...
  toc/60*(maxiter - i)));
    end
    tic
  end
end
```


A.4 Semi Definite $\mathbf{L}^T\mathbf{L}$

A.4.1 jacobicgsemi

Purpose Conjugate gradient method with a two-grid Jacobi-like regularized preconditioner with a semidefinite $\mathbf{L}$.

Synopsis [X, rho, eta, flopc] = jacobicgsemi(K,Vk,alpha,L,L0,b,maxiter,reg)

Description Performs maxiter iterations of conjugate gradients with a two-grid block diagonal preconditioner (Jacobi-like), either the unregularized normal equation system

$$\mathbf{K}^T\mathbf{K}\mathbf{x} = \mathbf{b} \qquad \text{if reg} \neq 1$$

or on the regularized system (default)

$$(\mathbf{K}^T\mathbf{K} + \alpha\mathbf{L}^T\mathbf{L})\mathbf{x} = \mathbf{K}^T\mathbf{b} \qquad \text{if reg} = 1$$

The preconditioner is formed from the subspace Vk, the ill-posed matrix K, the regularization matrix L, the null space of the regularization matrix L0 $= \mathcal{N}(\mathsf{L})$ and the regularization parameter alpha. The null space of K and L are not allowed to intersect.

The result of each iteration is returned in X and the norm of the residuals and solutions are found in rho and eta respectively. A flop-count from each iteration is stored in flopc.

If parameter trace is stated an estimation of remaining calculation time is printed after each iteration.

Code

```
function [X, rho, eta, flopc] =  ...
    jacobicgsemi(K,V,alpha,L,L0,b,maxiter,reg,trace)
% JACOBICG  Conjugated gradients with two-level Jacobi
%           like preconditioner for ill-posed problems.
%           (for semidefinite L)
%
% [X, rho, eta, flopc] =
%     jacobicgsemi(K,V,alpha,L,b,maxiter {{,reg},trace})
%
% Solves a ill-posed problem with or without tikhonov
```

```
% regularization, i.e
%
% (K'K + L'L)u = K'b              (1)
%    or
% K'K u = K'b                     (2)
%
% Input arguments:
%  K     : Kernel matrix
%  V     : Basis for coarse subspace
%  alpha : Regularization parameter used in preconditioner and
%          in (1).
%  L     : Seminorm matrix
%  L0    : Null space of L
%  b     : Right hand side vector
%  iter  : Maximum number of iterations
%  reg   :  =1 Use CG on (1)
%          ~=1 Use CG on (2)
%          Optional, default: 1
%  trace : Any value yields output information during
%          iterations; usefull for large problems.
%          Optional, default: off
%
% Output arguments:
%  X     : A matrix with all iter solutions stored as columns
%  rho   : A vector containing residuals
%
% References:
% [1] Michael Jacobsen: Two-Grid Iterative Methods
%                       for Ill-Posed Problems

% Last Revised: 2000.08.03

% Checking input arguments and setting defauls
identity = 0;
if nargin < 9
  trace = 0;
  if nargin < 8
    reg = 1;
  end
end
if isempty(L)
  identity = 1;
end
if nargout == 4
  flops(0);
end

% Prepare projection operators
nd = size(L0,2);
```



```
k = size(V,2);
V2 = V;
for i=1:k
  for j=1:nd
    V2(:,i) = V2(:,i) - (V2(:,i)'*L0(:,j))*(L0(:,j));
    V2(:,i) = V2(:,i) / norm(V2(:,i));
  end
end
V = [L0 V2];

% Initial work (factorizations etc.)
KV = K*V; LV = L*V;
A11  = KV'*KV + alpha*LV'*LV;
clear KV LV;

RA11 = chol(A11);
G = (V2'*L'*L*V2);
RG = chol(G);
RL = triu(qr(L));

b = K'*b;

% Allocate memory
u = zeros(size(K,2),1);
X = zeros(size(K,2),maxiter);
rho = zeros(maxiter,1);
r = b;
delta = 0;

for i=1:maxiter
  % Preconditioning step
  e1 = V*(RA11\(RA11'\(V'*r)));
  warning off
  w = RL\(RL'\r);
  warning on
  w = w - L0*(L0'*w);
  e2 = w /alpha;
  e3 = V2*(RG\(RG'\(V2'*r)))/alpha;
  z = e1 + e2 - e3;
  % Preconditioning step end

  % Cg step
  deltaold = delta;
  delta    = r'*z;
  if i == 1
    p = z;
  else
    beta = delta / deltaold;
    p = z + beta*p;
```

```
  end
  d = K'*(K*p);
  % Work on regularized system or not
  if reg == 1
    if identity
      d = d + alpha*p;             % Non regularized
    else
      d = d + alpha*(L'*(L*p));    % Regularized
    end
  end

  % Update data
  alfa = delta / (p'*d);
  u = u + alfa*p;
  X(:,i) = u;
  r = r - alfa*d;

  % Store information, if required
  if nargout > 1, rho(i) = norm(r); end
  if nargout > 2, eta(i) = norm(u); end
  flopc(i) = flops;

  % Print trace information
  if trace
    disp(['Iteration ', num2str(i), ' in progress'])
    if i > 1
      disp(sprintf('Done in %3.1f minutes', ...
  toc/60*(maxiter - i)));
    end
    tic
  end
end
```


A.4.2 gscgsemi

Purpose Conjugate gradient method with two-grid symmetric Gauss-Seidel-like regularized preconditioning.

Synopsis [X, rho, eta, flopc] = gscg(K,Vk,alpha,L,L0,b,iter{{,reg},trace})

Description Performs maxiter iterations of conjugate gradients with a two-grid block symmetric Gauss-Seidel-like preconditioner, either on the unregularized normal equation system

$$\mathbf{K}^T\mathbf{K}\mathbf{x} = \mathbf{b} \qquad \text{if reg} \neq 1$$

or on the regularized system (default)

$$(\mathbf{K}^T\mathbf{K} + \alpha\mathbf{L}^T\mathbf{L})\mathbf{x} = \mathbf{K}^T\mathbf{b} \qquad \text{if reg} = 1$$

The preconditioner is formed from the subspace Vk, the ill-posed matrix K, the regularization matrix L, the null space of the regularization matrix L0 $= \mathcal{N}(\mathsf{L})$ and the regularization parameter alpha. The null space of K and L must not intersect.

The result of each iteration is returned in X and the norm of each the residual and solution is found in rho and eta respectively.

If trace is stated an estimated remaining running time is displayed after each iteration.

Code

```
function [X,rho,eta,flopc] = ...
    gscg(K,V,alpha,L,b,iter,reg,trace)
% GSCG  Conjugated gradients with two-level preconditioner
%       Gauss-Seidell like preconditioner for ill-posed
%       problems.
%
% [X, rho, eta, flopc] =
%                GSCG(K,Vk,alpha,L,b,maxiter,reg)
%
% Solves a ill-posed problem with or without tikhonov
% regularization, i.e
%
% (K'K + L'L)u = K'b              (1)
%    or
```

```
% K'K u = K'b                      (2)
%
% using conjugated gradients with a two grid Gauss-Seidel
% like regularized precondtioner.
%
% Input arguments:
%  K     : Kernel matrix
%  V     : Basis for coarse subspace
%  alpha : Regularization parameter used in preconditioner
%          and in (1).
%  L     : Seminorm matrix (SPD)
%          Use [] for identity
%  b     : Right hand side vector
%  iter  : Maximum number of iterations
%  reg   :  =1 Use CG on (1)  (Default)
%          ~=1 Use CG on (2)
%  trace : Output estimated remaining time after each iteration
%          Default off
%
% Output arguments:
%  X     : A matrix with all iter solutions stored as columns
%  rho   : A vector containing residuals
%  flopsc: A flopcount after each iteration.
%
% References:
% [1] Michael Jacobsen: Two-Grid Iterative Methods
%                       for Ill-Posed Problems

% Last Revised: 2000.09.02

% Checking input arguments
identity = 0;
if nargin < 8
  trace = 0;
  if nargin < 7
    reg = 1;
  end
end
if isempty(L)
  identity = 1;
end
if nargout == 4
  flops(0);
end

% Initial work (factorizations etc.)
b = K'*b;
KV = K*V;
if identity
```



```
  A11 = KV'*KV + alpha*V'*V;
  clear KV;
else
  LV = L*V;
  A11 = KV'*KV+alpha*LV'*LV;
  clear KV LV;
end

R = chol(A11);
if identity
  G  = V'*V;
  GR = triu(qr(V));
else
  LV = L*V;
  G  = LV'*LV;
  GR = triu(qr(LV));
  LR = triu(qr(L));
  clear LV;
end


% Allocate memory
u = zeros(size(K,2),1);
X = zeros(size(K,2),iter);
rho = zeros(iter,1);
r = b;
delta = 0;

for i=1:iter
  % Preconditioning step
  v = V*(R\(R'\(V'*r)));
  e = r - K'*(K*v);
  if identity
    w = e;                                  % M\e;
  else
    w = LR\(LR'\e);                         % M\e;
  end

  px = w - V*(GR\(GR'\(V'*e)));             % V*G\V'*e
  uq = px / alpha;
  y = r - K'*(K*uq);
  up = V*(R\(R'\(V'*y)));
  z = up + uq;

  % Cg step
  deltaold = delta;
  delta    = r'*z;
  if i == 1
    p = z;
```

```
  else
    beta = delta / deltaold;
    p = z + beta*p;
  end
  d = K'*(K*p);
  % Work on regularized system or not
  if reg == 1
    if identity
      d = d + alpha*p;
    else
      d = d + alpha*(L'*(L*p));
    end
  end

  alfa = delta / (p'*d);
  u = u + alfa*p;
  X(:,i) = u;
  r = r - alfa*d;

  % Update norms, if required
  if nargout > 1, rho(i)   = norm(r); end
  if nargout > 2, eta(i)   = norm(u); end
  flopc(i) = flops;

  % Print trace information
  if trace
    disp(['Iteration ', num2str(i), ' in progress'])
    if i > 1
      disp(sprintf('Done in %3.1f minutes', ...
  toc/60*(iter - i)));
    end
    tic
  end
end
```


A.4.3 schurcgsemi

Purpose Schur complement conjugate gradient method with regularization for ill-posed problems.

Synopsis X = schurcgsemi(K,V,alpha,L,L0,b,maxiter{,trace})

Description Performs maxiter iterations of the Schur complement conjugate gradient method on the regularized system

$$(\mathbf{K}^T\mathbf{K} + \alpha\mathbf{L}^T\mathbf{L})\mathbf{x} = \mathbf{K}^T\mathbf{b} \tag{A.2}$$

The Schur complement CG operates on a subspace which is $\mathbf{L}^T\mathbf{L}$-orthogonal to the subspace defined by V and L0 (the null space of L). The null space of L and K must not intersect.

The result of each iteration is returned in X. The norm of the residuals and solutions are stored in rho and eta respectively.

If trace is stated an estimated remaining running time is displayed after each iteration.

Code

```
function [X, rho, eta, flopc] = ...
    schurcgsemi(K,V,alpha,L,L0,b,maxiter,trace)
% SCHURCGSEMI  Schur Complement Conjugated Gradients
%          Assumes L to be semidefinie. Nullspace
%          required.
%
%   [X, rho, eta, flopc] =
%                SCHURCGSEMI(K,V,alpha,L,W,b,maxiter)
%
% Solves a tikhonov regularized system of type
%       (K'K+L'L)u = K'*b
% using schur complement conjugated gradients -- see [1].
%
% Input arguments:
%  K     : Kernel matrix
%  V     : Basis for coarse subspace (L0 is added)
%  alpha : Regularization parameter
%  L     : Seminorm matrix
%  L0    : Null space of L
%  b     : Right hand side vector
%  iter  : Maximum number of iterations
%  trace : Print estimated remaining run time after
%          each iteraiton
```

```
%          Default off
%
% Output argument:
%  X     : A matrix with all iter solutions stored as columns
%  rho   : Residual norm from each iteration
%  eta   : Solution norm form each iteration
%  flopc : Flop count from each iteration
%
% References:
% [1] Michael Jacobsen: Two-Grid Iterative Methods
%                       for Ill-Posed problems

% Last Revised: 2000.09.02

if nargin < 9
  trace = 0;
end
if nargout == 4
  flops(0);
end

% Prepare projection operators
p = size(L0,2);
k = size(V,2);
V2 = V;
for i=1:k
  for j=1:p
    V2(:,i) = V2(:,i) - (V2(:,i)'*L0(:,j))*(L0(:,j));
    V2(:,i) = V2(:,i) / norm(V2(:,i));
  end
end
V = [L0 V2];

% Cholesky factorization of K'K
KV  = K*V; LV  = L*V;
A11 = KV'*KV+alpha*LV'*LV;
G   = L*V2; G   = G'*G;

RA11 = chol(A11);
RG   = chol(G);
RL   = triu(qr(L));

% We are working with normal equations
b = K'*b;

% Initialization
x = V*(RA11\(RA11'\(V'*b)));
r = b - K'*(K*x) - alpha*(L'*(L*x));
% Turn off warnings
```



```
warning off
q = RL\(RL'\r);
warning on
y = q - V2*(RG\(RG'\(V2'*r))) - L0*(L0'*q);
z = r;
d = y;
delta0 = y'*r;


for i=1:maxiter
  tt = K'*(K*d) + alpha*(L'*(L*d));        % A*z
  v  = d - V*(RA11\(RA11'\(V'*tt)));
  w  = K'*(K*v) + alpha*(L'*(L*v));        % A*v
  warning off
  f  = RL\(RL'\w);
  warning on
  g  = f - V2*(RG\(RG'\(V2'*w))) - L0*(L0'*f);
  tau = delta0 / (z'*g);
  x = x + tau*v;
  X(:,i) = x;                            % Store solution
  r = r - tau*w;
  y = y - tau*g;
  delta1= r'*y;
  beta = delta1 / delta0;
  delta0 = delta1;
  z = r + beta*z;
  d = y + beta*d;

  % Store norm, if required
  if nargout > 1, rho(i) = norm(r); end
  if nargout > 2, eta(i) = norm(u); end
  flopc(i) = flops;

  % Print trace information
  if trace
    disp(['Iteration ', num2str(i), ' in progress'])
    if i > 1
      disp(sprintf('Done in %3.1f minutes', ...
  toc/60*(maxiter - i)));
    end
    tic
  end
end
```

A.5 Wavelet Schur complement

A.5.1 schurcgwavelet

Purpose Schur complement conjugate gradient method for ill-posed problems. Subspace splitting with wavelets.

Synopsis [X, rho, eta, flopc] = schurcg(K,D,k,alpha,L,b,maxiter{,trace})

Description Performs maxiter iterations of the Schur complement conjugate gradient method on the regularized system

$$(\mathbf{K}^T\mathbf{K} + \alpha\mathbf{L}^T\mathbf{L})\mathbf{x} = \mathbf{K}^T\mathbf{b} \tag{A.3}$$

The subspace splitting is done by a Daubechies wavelet basis with genus D and dimension k.

See also schurcg.

Code

```
function [X, rho, eta, flopc] = ...
    schurcgwavelet(K,D,k,alpha,L,b,maxiter,trace)
% SCHURCGWAVELET  Schur Complement Conjugated Gradients
%                 using a wavelet basis
%                 Assumes L to be invertible.
%
%   [X, rho, eta, flopc] =
%              SCHURCGWAVELET(K,D,k,alpha,L,b,maxiter {,trace})
%
% Solves a tikhonov regularized system of type
%        (K'K+alpha*L'L)u = K'*b
% using Schur complement conjugated gradients -- see [1].
% The subspace projections are done with fast wavelet transforms
%
% Ole Moller Nielsen's wavelet package is required:
% http://www.imm.dtu.dk/~omni/wt.html
%
% Input arguments:
%  K     : Kernel matrix
%  D     : Wavelet Genus
%  k     : Subspace dimension
%  alpha : Regularization parameter
%  L     : Seminorm matrix (SPD)
%          Use [] for identity
%  b     : Right hand side vector
```



```
%  iter  : Maximum number of iterations
%  trace : Print progress report after each iteration.
%
% Output argument:
%  X     : A matrix with all iter solutions stored as columns
%  rho   : Residual norms
%  eta   : Solution norms
%  flopc : Flopcount after each iteration.
%
% References:
% [1] Michael Jacobsen: Two-Grid Iterative Methods
%                       for Ill-Posed problems

% Last Revised: 2000.09.02

global WD SK N Vk;

if nargin < 8
  trace = 0;
end
if nargout == 4
  flops(0);
end

N = length(b);
SK = k; WD = D;

% Cholesky factorization of K'K and L'L
if ~isempty(L)
  KV  = fwt(K',D); LV  = fwt(L',D);
  A11 = KV(1:k,:)*KV(1:k,:)' + alpha*LV(1:k,:)*LV(1:k,:)';
  RL  = triu(qr(L));
else
  KV  = fwt(K',D);
  A11 = KV(1:k,:)*KV(1:k,:)' + alpha*eye(k);
  RL  = speye(N);
end
RA11 = chol(A11);

% We are working with normal equations
b = K'*b;

% Initialization
u = prolong(RA11\(RA11'\(restrict(b))));
if ~isempty(L)
  r = b - K'*(K*u) - alpha*(L'*(L*u));
  y = RL\(RL\r);
else
  r = b - K'*(K*u) - alpha*u;
```

```
  y = r;
end

d = r;
z = y;
delta0 = y'*r;
X = [];

for i=1:maxiter
  if ~isempty(L)
    tt = K'*(K*z) + alpha*(L'*(L*z));         % A*z
  else
    tt = K'*(K*z) + alpha*z;                  % A*z
  end
  v  = z - prolong(RA11\(RA11'\(restrict(tt))));
  if ~isempty(L)
    w  = K'*(K*v) + alpha*(L'*(L*v));         % A*v
    g  = RL\(RL'\w);
  else
    w  = K'*(K*v) + alpha*v;                  % A*v
    g  = w;
  end
  tau = delta0 / (d'*g);
  u = u + tau*v;
  X(:,i) = u;                      % Store solution
  r = r - tau*w;
  y = y - tau*g;
  delta1= r'*y;
  beta = delta1 / delta0;
  delta0 = delta1;
  z = y + beta*z;
  d = r + beta*d;

  % Store iteration data, if required
  if nargin > 1, rho(i) = norm(r); end
  if nargin > 2, eta(i) = norm(u); end
  flopc(i) = flops;
end

function wo = restrict(wi)
global SK WD;
wo = fwt(wi,WD);
wo = wo(1:SK);

function o = prolong(w)
global WD N SK;
o = zeros(N,1);
o(1:SK) = w;
o = ifwt(o,WD);
```


A.6 2-D Schur Complement

A.6.1 schurcg2d

Purpose Schur complement conjugate gradient method for ill-posed problems with Kronecker product kernels.

Synopsis [X, rho, eta, flopc] = schurcg2d(K1,K2,V1,V2,alpha,L1,L2,B,novev,maxiter)

Description Performs maxiter iterations of the Schur complement conjugate gradient method on the regularized system

$$(\mathbf{K}^T\mathbf{K} + \alpha\mathbf{L}^T\mathbf{L})\mathrm{vec}(\mathbf{x}) = \mathbf{K}^T\mathrm{vec}(b), \tag{A.4}$$

where $\mathbf{K} = \mathbf{K}_1 \otimes \mathbf{K}_2$, $\mathbf{V} = \mathbf{V}_1 \otimes \mathbf{V}_2$, and $\mathbf{L} = \mathbf{L}_1 \otimes \mathbf{L}_2$. The algorithms uses matrix-matrix products according to the algebraic rules of Kronecker products. The parameter novec tells the approximate $\mathbf{A}_{11}^{-1}$ with a Kronecker product or not.

See also schurcg.

Code

```
function [X, rho, eta, flopc] = ...
    schurcg2d(K1,K2,V1,V2,alpha,L1,L2,B,maxiter,novec,trace)
% SCHURCG2D  Schur Complement Conjugated Gradients on
%            Kronecker product problem
%            Assumes L to be invertible.
%
%   [X, rho, eta, flopc] =
%       schurcg2d(K1,K2,V1,V2,alpha,L1,L2,b,maxiter {,trace})
%
% Solves a tikhonov regularized system of type
%       (K'K+L'L)u = K'*b
% using schur complement conjugated gradients -- see [1].
%
% Input arguments:
%  K1    : Kernel matrix
%  K2    : Kernel matrix
%  V1    : Basis for coarse subspace
%  V2    : Basis for coarse subspace
%  alpha : Regularization parameter
%  L1    : Seminorm matrix
%  L2    : Seminorm matrix
```

```
%  B     : Right hand side matrix
%  iter  : Maximum number of iterations
%  novec : Use only matrix approximation
%  trace : (Optional) Print information during calculations
%
% Output argument:
%  X     : A matrix with all iter solutions stored as columns
%  rho   : Residual norms
%  eta   : Solution norms
%  flopc : Flop counts
%
% References:
% [1] Michael Jacobsen: Two-Grid Iterative Methods
%                       for Ill-Posed problems

% Last Revised: 2000.09.02

if nargout == 4
  flops(0);
end
if nargin < 11
  trace = 0;
end

SV1 = size(V1,2); SV2 = size(V2,2);
SK1 = size(K1,2); SK2 = size(K2,2);

% Cholesky factorization of K'K and L'L
KV1 = K1*V1; KV2 = K2*V2;
LV1 = L1*V1; LV2 = L2*V2;
if ~novec
  A11 = kron(KV1'*KV1,KV2'*KV2) + ...
      alpha*kron(LV1'*LV1,LV2'*LV2);
  RA11 = chol(A11);
else
  A1r = chol(KV1'*KV1);
  A2r = chol(KV2'*KV2);
end
RL1 = qr(L1); RL2 = qr(L2);

% A11 is SPD (at least theoreticaly)
if trace
  disp('A11 done')
end

% We are working with normal equations
B = K2'*B*K1;

% Initialization
```



```
if novec
  U = V2'*B*V1;
  U = (A2r\(A2r'\U))/A1r/A1r';
  U = V2*U*V1';
else
  U = reshape(V2'*B*V1,SV1*SV2,1);      % To vector
  U = reshape(RA11\(RA11'\U),SV2,SV1); % To matrix
  U = V2*U*V1';
end
R = B - K2'*K2*U*K1'*K1 - alpha*L2'*L2*U*L1'*L1;

Y = R;
D = R;
Z = Y;

delta0 = Y(:)'*R(:);
X = zeros(SK2,SK1,maxiter);

for i=1:maxiter
  TT = K2'*K2*Z*K1'*K1 + alpha*L2'*L2*Z*L1'*L1;
  if novec
    TT2  = V2'*TT*V1;
    TT3 = (A2r\(A2r'\TT2))/A1r/A1r';
  else
    TT2  = reshape(V2'*TT*V1,SV2*SV1,1);
    TT3  = reshape(RA11\(RA11'\(TT2)),SV1,SV2);
  end
  V = Z - V2*TT3*V1';

  W = K2'*K2*V*K1'*K1 + alpha*L2'*L2*V*L1'*L1;
  G = W;

  tau = delta0 / (D(:)'*G(:));
  U = U + tau*V;
  X(:,:,i) = U;                         % Store solution
  R = R - tau*W;                        % Update residual
  Y = Y - tau*G;
  delta1= R(:)'*Y(:);
  beta = delta1 / delta0;
  delta0 = delta1;
  Z = Y + beta*Z;
  D = R + beta*D;

  % Store information, if required
  if nargout > 1, rho(i) = norm(R,'fro'); end
  if nargout > 2, eta(i) = norm(U,'fro'); end
  flopc(i) = flops;

  % Print trace information
```

```
  if trace
    disp(['Iteration ', num2str(i), ' in progress'])
    if i > 1
      disp(sprintf('Done in %3.1f minutes', ...
  toc/60*(maxiter - i)));
    end
    tic
  end
end
```

Appendix B

Proofs

This part of the appendix contains a proof of an equality. It is ommited from the main text because it was irrelevant in the overall context, but included here because the equality is not obvious.

B.1 Jacobi Preconditioning

We have a matrix $\mathbf{T} \in \mathbb{R}^{n \times k}$ and it's *thick* SVD

$$\mathbf{T} = \mathbf{U} \begin{bmatrix} \mathbf{\Sigma} \\ \mathbf{0}_{n-k,k} \end{bmatrix} \mathbf{V}^T$$

where $\mathbf{\Sigma}$ and $\mathbf{V}$ equals those of the usual thin SVD. The first columns of the thick SVD $\mathbf{U}$ equals the columns the usual thin SVD. But the thick SVD enlarges $\mathbf{U}$ with columns spanning the first columns orthogonal complement. The result is that $\mathbf{U}$ is square and orthogonal.

The objective is to prove

$$\left(\mathbf{T}^T\mathbf{T} + \alpha \mathbf{I}_k\right)^{-1} \mathbf{T}^T = \mathbf{T}^T \left(\mathbf{T}\mathbf{T}^T + \alpha \mathbf{I}_n\right)^{-1} \tag{B.1}$$

The first step inserts the thick SVD of $\mathbf{T}$ at all occurances in the left hand

side of (B.1)

$$\left(\mathbf{V} \begin{bmatrix} \mathbf{\Sigma} & \mathbf{0}_{k,n-k} \end{bmatrix} \mathbf{U}^T\mathbf{U} \begin{bmatrix} \mathbf{\Sigma} \\ \mathbf{0}_{n-k,k} \end{bmatrix} \mathbf{V}^T + \alpha \mathbf{I}_k\right)^{-1} \mathbf{V} \begin{bmatrix} \mathbf{\Sigma} & \mathbf{0}_{k,n-k} \end{bmatrix} \mathbf{U}^T$$
$$=$$
$$\left(\mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^T + \alpha \mathbf{I}_k\right)^{-1} \mathbf{V} \begin{bmatrix} \mathbf{\Sigma} & \mathbf{0}_{k,n-k} \end{bmatrix} \mathbf{U}^T$$
$$=$$
$$\left(\mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^T + \alpha \mathbf{V}\mathbf{I}_k\mathbf{V}^T\right)^{-1} \mathbf{V} \begin{bmatrix} \mathbf{\Sigma} & \mathbf{0}_{k,n-k} \end{bmatrix} \mathbf{U}^T$$

Secondly we move all $\mathbf{V}$s out of $(\ldots)^{-1}$

$$\mathbf{V}\left(\mathbf{\Sigma}^2 + \alpha \mathbf{I}_k\right)^{-1} \mathbf{V}^T\mathbf{V} \begin{bmatrix} \mathbf{\Sigma} & \mathbf{0}_{k,n-k} \end{bmatrix} \mathbf{U}^T$$
$$=$$
$$\mathbf{V}\left(\mathbf{\Sigma}^2 + \alpha \mathbf{I}_k\right)^{-1} \begin{bmatrix} \mathbf{\Sigma} & \mathbf{0}_{n-k,k} \end{bmatrix} \mathbf{U}^T$$

Now we multiply the two terms in the middle

$$\mathbf{V} \begin{bmatrix} \left(\mathbf{\Sigma}^2 + \alpha \mathbf{I}_k\right)^{-1} \mathbf{\Sigma} & \mathbf{0}_{k,n-k} \end{bmatrix} \mathbf{U}^T$$

and because diagonal matrices are commutable with respect to multiplication, i.e $\mathbf{\Sigma}_1\mathbf{\Sigma}_2 = \mathbf{\Sigma}_2\mathbf{\Sigma}_1$, we are able to commute the terms $(\ldots)^{-1}$ and $\mathbf{\Sigma}$ and write the block matrix as the sum of a matrix multiplication (note the lower right block)

$$\mathbf{V} \begin{bmatrix} \mathbf{\Sigma}\left(\mathbf{\Sigma}^2 + \alpha \mathbf{I}_k\right)^{-1} & \mathbf{0}_{n-k,k} \end{bmatrix} \mathbf{U}^T$$
$$=$$
$$\mathbf{V} \begin{bmatrix} \mathbf{\Sigma} & \mathbf{0}_{k,n-k} \end{bmatrix} \begin{bmatrix} \left(\mathbf{\Sigma}^2 + \alpha \mathbf{I}_k\right)^{-1} & \mathbf{0}_{k,n-k} \\ \mathbf{0}_{n-k,k} & 1/\alpha \mathbf{I}_{n-k} \end{bmatrix} \mathbf{U}^T$$

The inverse of a block diagonal matrix equals the matrix of the inverse blocks

$$\mathbf{V}\begin{bmatrix}\boldsymbol{\Sigma} & \mathbf{0}_{k,n-k}\end{bmatrix}\begin{bmatrix}\boldsymbol{\Sigma}^2+\alpha\mathbf{I}_k & \mathbf{0}_{k,n-k}\\ \mathbf{0}_{n-k,k} & \alpha\mathbf{I}_{n-k}\end{bmatrix}^{-1}\mathbf{U}^T$$
$$=$$
$$\mathbf{V}\begin{bmatrix}\boldsymbol{\Sigma} & \mathbf{0}_{k,n-k}\end{bmatrix}\left(\begin{bmatrix}\boldsymbol{\Sigma}^2 & \mathbf{0}_{k,n-k}\\ \mathbf{0}_{n-k,k} & \mathbf{0}_{n-k}\end{bmatrix}+\alpha\mathbf{I}_n\right)^{-1}\mathbf{U}^T$$
$$=$$
$$\mathbf{V}\begin{bmatrix}\boldsymbol{\Sigma} & \mathbf{0}_{k,n-k}\end{bmatrix}\mathbf{U}^T\mathbf{U}\left(\begin{bmatrix}\boldsymbol{\Sigma}^2 & \mathbf{0}_{k,n-k}\\ \mathbf{0}_{n-k,k} & \mathbf{0}_{n-k}\end{bmatrix}+\alpha\mathbf{I}_n\right)^{-1}\mathbf{U}^T\mathbf{U}\mathbf{U}^T$$
$$=$$
$$\mathbf{V}\begin{bmatrix}\boldsymbol{\Sigma} & \mathbf{0}_{k,n-k}\end{bmatrix}\mathbf{U}^T\left(\mathbf{U}\begin{bmatrix}\boldsymbol{\Sigma}^2 & \mathbf{0}_{k,n-k}\\ \mathbf{0}_{n-k,k} & \mathbf{0}_{n-k}\end{bmatrix}\mathbf{U}^T+\alpha\mathbf{I}_n\right)^{-1} \tag{B.2}$$

A computation of $\mathbf{T}\mathbf{T}^T$ using the thick SVD reveals

$$\begin{aligned}\mathbf{T}\mathbf{T}^T &= \mathbf{U}\begin{bmatrix}\boldsymbol{\Sigma}\\ \mathbf{0}_{n-k,k}\end{bmatrix}\mathbf{V}^T\mathbf{V}\left[\boldsymbol{\Sigma}\ \mathbf{0}_{k,n-k}\right]\mathbf{U}^T\\ &= \mathbf{U}\begin{bmatrix}\boldsymbol{\Sigma}\\ \mathbf{0}_{n-k,k}\end{bmatrix}\left[\boldsymbol{\Sigma}\ \mathbf{0}_{k,n-k}\right]\mathbf{U}^T\\ &= \mathbf{U}\begin{bmatrix}\boldsymbol{\Sigma} & \mathbf{0}_{k,n-k}\\ \mathbf{0}_{n-k,k} & \mathbf{0}_{n-k}\end{bmatrix}\mathbf{U}^T\end{aligned}$$

which combined with (B.2) yields the desired result

$$\mathbf{V}\begin{bmatrix}\boldsymbol{\Sigma} & \mathbf{0}_{k,n-k}\end{bmatrix}\mathbf{U}^T\left(\mathbf{U}\begin{bmatrix}\boldsymbol{\Sigma}^2 & \mathbf{0}_{k,n-k}\\ \mathbf{0}_{n-k,k} & \mathbf{0}_{n-k,n-k}\end{bmatrix}\mathbf{U}^T+\alpha\mathbf{I}_n\right)^{-1}$$
$$=$$
$$\mathbf{T}^T\left(\mathbf{T}\mathbf{T}^T+\mathbf{I}_n\right)^{-1}$$

$\square$

Bibliography

[1] O. AXELSSON, *Iterative Solution Methods*, Cambridge University Press, 1994.

[2] R. BARRET ET AL., *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM, Philadelphia, 1994.

[3] W. L. BRIGGS, *A Multigrid Tutorial*, SIAM, Philadelphia, 1987.

[4] I. DAUBECHIES, *Ten Lectures on Wavelets*, SIAM, Philadelphia, 1992.

[5] J. W. DEMMEL, *Applied Numerical Linear Algebra*, SIAM, Philadelphia, 1997.

[6] M. FEDI AND A. RAPOLLA, *3-D inversion of gravity and magnetic data with depth resolution*, Geophysics, 64 (1999), pp. 452–460.

[7] G. H. GOLUB AND C. F. V. LOAN, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, Maryland, third ed., 1996.

[8] G. H. GOLUB AND D. P. O'LEARY, *History of the conjugate gradient and Lanczos methods*, SIAM Review, 31 (1989), pp. 50–102.

[9] A. GREENBAUM, *The Lanczos and conjugate gradient algorithms in finite precision arithmetic*, in Proceedings of the Cornelius Lanczos International Centenary Conference, SIAM, 1994, pp. 49–60.

[10] L. A. HAGEMAN AND D. M. YOUNG, *Applied Iterative Methods*, Computer Science and Applied Mathematics, Academic Press, New York, 1981.

[11] M. HANKE, *Conjugate Gradient Type Methods for Ill-Posed Problems*, Pitman Research Notes in Mathematics Series, Longman Group Limited, Harlow, England, 1995.

[12] M. HANKE AND C. R. VOGEL, *Two-level preconditioners for regularized inverse problems II: Implementation and numerical results*. Submitted to SIAM Journal on Scientific Computing.

[13] ——, *Two-level preconditioners for regularized inverse problems I: Theory*, Numerische Mathematik, 83 (1999), pp. 385–402.

[14] P. C. HANSEN, *Deconvolution and regularization with Toeplitz matrices*. Submitted to Numerical Algorithms.

[15] ——, *The discrete Picard condition for discrete ill-posed problems*, BIT, (1990), pp. 658–672.

[16] ——, *Rank-Deficient and Discrete Ill-Posed Problems*, SIAM, Philadelphia, 1998.

[17] ——, *Regularization tools version 3.0 for Matlab 5.2*, Numerical Algorithms, 20 (1999), pp. 195–196.

[18] M. R. HESTENES AND E. STIEFEL, *Methods of conjugate gradients for solving linear systems*, Journal of Research of the National Bureau of Standards, 49 (1952), pp. 409–436.

[19] M. JACOBSEN, J. M. RASMUSSEN, AND H. SØRENSEN, *Image-restoration using PP-TSVD*, 1998.

[20] J. T. KING, *Multilevel algorithms for ill-posed problems*, Numerische Mathematik, 61 (1992), pp. 311–334.

[21] C. V. LOAN, *Computational Frameworks for the Fast Fourier Transform*, Frontiers in Applied Mathematics, SIAM, Philadelphia, 1992.

[22] A. NEUMAIER, *Solving ill-conditioned and singular linear systems: A tutorial on regularization*, SIAM Review, 40 (1998), pp. 636–666.

[23] O. M. NIELSEN, *Wavelets in Scientific Computing*, PhD thesis, Technical University of Denmark, July 1998.

[24] C. C. PAIGE AND M. A. SAUNDERS, *LSQR: Sparse linear equations and least squares problems*, ACM Transactions on Mathematical Software, 8 (1982), pp. 195–209.

[25] B. N. PARLETT, *The Symmetric Eigenvalue Problem*, Classics in applied mathematics, SIAM, Philadelphia, 1998.

[26] M. PEDERSEN, *Functional Analysis in Applied Mathematics and Engineering*, Chapman & Hall/CRC, Boca Ranton, Florida, 1999.

[27] D. L. PHILLIPS, *A technique for the numerical solution of certain integral equations of the first kind*, Journal of ACM, 9 (1962), pp. 84–97.

[28] A. RIEDER, *A wavelet multilevel method for ill-posed problems stabilized by Tikhonov regularization*, Numerische Mathematik, 75 (1997), pp. 501–522.

[29] K. L. RILEY, *Two-Level Preconditioners for Regularized Ill-Posed Problems*, PhD thesis, Montana State University – Bozeman, July 1999.

[30] Y.-H. D. ROECK, *Sparse linear algebra and geophysical migration*, Tech. Rep. RR-3876, INRIA Rennes, February 2000.

[31] J. R. SHEWCHUK, *An introduction to the conjugate gradient method with-*

out the agonizing pain. http://www.cs.cmu.edu/~quake-papers/, August 1994.

[32] B. SMITH, P. BJØRSTAD, AND W. GROPP, *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*, Cambridge University Press, 1996.

[33] A. N. TIKHONOV AND V. Y. ARSENIN, *Solutions of Ill-Posed Problems*, Scripta Series in Mathematics, John Wiley & Sons, New York, 1977.

[34] L. N. TREFETHEN AND D. B. III, *Numerical Linear Algebra*, SIAM, Philadelphia, 1997.

[35] C. R. VOGEL, *Negative results for multilevel preconditioners in image deblurring*, in Scale-Space Theories in Computer Vision, M. Nielsen et al., eds., Springer, 1999.