Mater of Science Thesis
Literature Study

Improving the performance of the VISAGE linear solver for coupled IX-VISAGE simulation in context of CCS

Arnoud Glasbeek

Faculty of Electrical Engineering, Mathematics and Computer Science

Delft University of Technology

Improving the performance of the VISAGE linear solver for coupled IX-VISAGE simulation in context of CCS

Literature Study

submitted in partial fulfillment of the
requirements for the degree of

Master of Science

in

Applied Mathematics

by

Arnoud Glasbeek,
Student number 5122473,
born in Gouda, The Netherlands

This work was performed in:

Numerical Analysis Group
Department of Delft Institute of Applied Mathematics
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

In collaboration with:

Abingdon Technology Center
Schlumberger Oilfield UK Ltd. (SLB)

Supervised by:

C. Vuik, Delft University of Technology
G. Isotton, SLB
A. Pearce, SLB
T. Jonsthovel, SLB

Delft University of Technology

Abstract

Carbon Capture and Storage is a potential solution for the reduction of CO_2 emission, that makes use of injecting CO_2 in geological formations. To establish the risks of gas injection in these formations, it is desired to accurately simulate its effects. For the simulation of this process, the Visage geomechanics simulator is an affected tool that can be used. A potential improvement to the Visage geomechanics simulator lies in the way in which linear systems describing the geomechanical process are solved. This report provides an overview of methods that can potentially be useful in improving the process of solving linear systems within the Visage geomechanics simulator.

Contents

List of Figures

List of Tables

Table of Acronyms

Acronym	Description	First Occurrence
AMG	Algebraic Multigrid	Section 4.5.3
BiCGSTAB	Biconjugate Gradient Stabilised	Section 4.2.4
CCS	Carbon Capture and Storage	Chapter 1
CG	Conjugate Gradient	Chapter 1
CR	Conjugate Residual	Section 4.2.4
CO_2	Carbon Dioxide	Chapter 1
FEM	Finite Element Method	Chapter 1
GMRES	Generalized Minimum Residual Method	Chapter 1
ILU	Incomplete LU	Section 5.4
ILUT	Incomplete LU with Threshold(s)	Section 5.4
PDE	Partial Differential Equation	Section 1
P-GMRES	Preconditioned GMRES	Section 5.2
SMG	Standard Multigrid	Section 4.5.2
SOR	Successive Over Relaxation	Section 4.1.4
SPD	Symmetric Positive Definite	Section 2.2
SSOR	Symmmetric Successive Over Relaxation	Section 4.1.4

Summary of acronyms used throughout this report, together with what their expansion and the section in which they are first used.

Table of Symbols

Symbol	Description
Bold letters ($\mathbf{x}, \mathbf{y}$, etc.)	(Generally) used for vectors, $\mathbf{x} \in \mathbb{R}^n$ denotes vector of size n.
Capital letters (A, B, etc.)	(Generally) used for matrices, $A \in \mathbb{R}^{m \times n}$ denotes matrix of size m by n.
Brackets with subscript of small case, E.g. $[A]_{i,j}$	Used to denote specific elements of vectors and matrices. $[A]_{i,j}$ denotes the element of matrix A in row i and column j
Brackets with subscript of two letters with :, E.g. $[A]_{i:j,k:l}$	Used to denote blocks of elements of vectors and matrices. $[A]_{i:j,k:l}$ denotes the block of elements of matrix A from row i to row j and from column k to column l
Brackets with subscript of capital letter E.g. $[A]_{S,T}$	Used to denote blocks of elements of vectors and matrices. $[A]_{S,T}$, with $S, T \subset \{1, \dots n\}$, denotes the block of elements of matrix A in rows i for $i \in S$ and columns j for $j \in T$
$\mathbf{0}_n$	Vector of size n with elements all 0
$\mathbf{1}_n$	Vector of size n with elements all 1
$\mathbf{e}_j$	Standard basis vector, consisting of all but one 0's and one 1 in position j
$O_{n,m}$	Zero matrix of sizes n by m
$(\mathbf{x}, \mathbf{y})$	Inner product of vectors, if not specified Euclidean inner product $(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T\mathbf{y}$ is used
$\|\mathbf{x}\|$	Norm of vector , if not specified Euclidean norm $\sqrt{(\mathbf{x}, \mathbf{x})}$ is used
$\|A\|$	Norm of matrix, if not specified Euclidean norm $\sup_{x \neq 0} \frac{\|Ax\|_2}{\|x\|_2}$ is used
$\lambda_i(A), \lambda_{\max}(A), \lambda_{\min}(A)$	i'th, maximum and minimum eigenvalue of matrix A
$\sigma(A)$	Spectrum of A, set of all eigenvalues
$\kappa(A)$	Condition number of A, given by $\kappa(A) = \|A\|\|A^{-1}\|$

Summary of usage of symbols throughout this report.

1 Introduction

One of the main challenges being faced right now is the reduction of emission of greenhouse gasses, particularly carbon dioxide (CO_2). The reduction of CO_2 emission is essential to combat rising trends in the global average temperature and the projected rise in sea level. As stated by Metz et al. **(2005 [6])**, several options to reduce the emission of CO_2 have been identified, such as reducing the use of fossil fuels by replacing the use of fossil fuel by renewable alternatives and enhancing the absorption of CO_2 by natural systems. Another option has been identified, which can specifically be applied to emission arising from industrial processes, referred to as Carbon Capture and Storage (CCS).

Certain processes, such as the combustion of fuels for power generation and production of hydrogen, steel and ammonia, can not be performed without CO_2 being produced. CSS technology aims to store the produced CO_2 instead of emitting it in the atmosphere. For this end, CO_2 is separated from other gasses produced during these processes and transported to a storage site, where it is supposed to remain for a long time **(Metz et al., 2005 [6])**.

One way of storing CO_2 is by injecting the gas into geological formations that contain hydrocarbons, such as abandoned oil or gas fields and unwinnable coal seams. The porosity and permeability make these formations suitable locations for the storage of CO_2-gas **(Design, 2010 [3])**. The injection of CO_2 in these formations however is not risk free. Leakage of the gas can lead to high CO_2 concentrations in the areas where the gas is injected, having significant negative effects on low-level ecosystems and potentially human health. It is thus desired to predict risks of collapses and leakages, which is done by simulating the evolution of the storage reservoirs **(Pires et al., 2011 [7])**.

To simulate the development of reservoirs for CSS, a mathematical model has to be constructed that takes into account both the reservoir itself and the mechanics associated with the reservoir. Using principles derived from geomechanics, a mathematical model consisting of Partial Differential Equations (PDEs) can be constructed. What this model exactly looks like heavily depends on the type of reservoir that is used. Using discretization methods, such as the Finite Element Method (FEM), a numerical model can be composed that is used to solve the PDEs derived for the monitoring of the storage reservoir.

An important tool for the simulation of reservoirs for CSS is the Petrel geomechanics module, developed by SLB **(SLB, 2024 [10])**, which provides a user-interface for generation and result viewing of simulations in reservoir engineering. For the FEM discretization, Petrel makes use of the Visage finite element geomechanics simulator, developed and innovated by SLB **(SLB, 2024 [11])**, with Visage performing calculations on rock stress and strain, displacements and failures.

Using the stress and strain, internal forces and displacements at a certain moment in time, Visage is able to determine these components at a later moment in time. To do this, large linear systems are required to be solved, which can be very costly in terms of computations, and thus in time required to develop a simulation. It is desired to find ways to make finding the solution to these linear systems computationally less costly without losing accuracy. The solution for this is expected to lie in a combination of using iterative solvers and applying methods that alter the way in which the FEM discretization is done.

Saad **(2003 [8])** discusses a wide range of iterative solvers for linear systems and is used to

derive the information on these iterative solvers discussed here. The iterative solvers discussed range from basic iterations methods to the more complicated Krylov subspace based methods. Basic iteration methods, such as Jacobi and Gauss-Seidel iteration, which make use of a certain iteration step, which is applied until a solution satisfies a chosen convergence condition. Methods based on projections on Krylov subspaces, such as Conjugate Gradient (CG) and Generalized Minimum Residual (GMRES), try to find a solution to a linear system by approximating the inverse of the matrix in the linear system by a polynomial of this matrix. These iterative solvers can further be improved using for example multigrid methods and preconditioning.

In this report, it is first discussed what the observed linear systems are and how they are obtained in Chapter 2. Next, in Chapter 3 methods to solve a linear system using a direct method, such as by computing the inverse or through Gaussian elimination, are discussed. Chapter 4 discusses several iterative methods, such as CG and GMRES, and principles that can be used to improve the computational efficiency of these methods, such as multigrid methods. Lastly, Chapter 5 discusses several methods to apply preconditioning to the earlier discussed iterative methods.

Observed Model

2

The primary use of finite element simulation observed here is the modelling of body forces in soil or rock formations. These forces are denoted, at a certain point in time t and in a finite amount of discretized integration points, by $\mathbf{p}^{(t)}$. It is desired to find a way to compute the force at a later time point, $t+\Delta t$. Apart from the forces at a certain point in time, the stresses, denoted by $\boldsymbol{\sigma}^{(t)}$, strains, denoted by $\boldsymbol{\epsilon}^{(t)}$, and displacements are known or can be determined from the forces, denoted by $\mathbf{u}^{(t)}$. So, it is desired to describe a function $f(\cdot)$, such that

$$\mathbf{p}^{(t+\Delta t)} = f(\mathbf{p}^{(t)}, \mathbf{u}^{(t)}, \boldsymbol{\sigma}^{(t)}, \boldsymbol{\epsilon}^{(t)}).$$

The stress and strain at a certain time can be described using the nodal displacement, which can be obtained from the forces at time t, by solving

$$K\mathbf{u}^{(t)} = \mathbf{p}^{(t)}, \tag{2.1}$$

where K denotes a stiffness matrix obtained from a finite element method. For this research, this last linear system is of most interest.

This chapter first discusses how a mathematical model that models the stress and strain in soil or rock formations is obtained in Section 2.1. After this it is discussed, in Section 2.2, how a system of the form of Equation (2.1) can be obtained using the model for stress and strain using the Finite Element Method.

2.1 Stress and Strain

To derive a model used to define an equation of the form of Equation (2.1), it is necessary to derive a way to obtain the stress and strain. Models observed here are three dimensional, which means that the stress, $\boldsymbol{\sigma}$, and strain, $\boldsymbol{\epsilon}$, consist of 6 components, each denoting stress or strain in a certain direction. In a certain location (x_i, y_i, z_i), the stress and strain can be described, each consisting of normal and shear components in each direction. This gives, in a location i, $\boldsymbol{\sigma}^i, \boldsymbol{\epsilon}^i \in \mathbb{R}^6$ as

$$\boldsymbol{\sigma}^i = \begin{pmatrix} \sigma^i_{xx} \\ \sigma^i_{yy} \\ \sigma^i_{zz} \\ \sigma^i_{xy} \\ \sigma^i_{yz} \\ \sigma^i_{zx} \end{pmatrix}, \qquad \boldsymbol{\epsilon}^i = \begin{pmatrix} \epsilon^i_{xx} \\ \epsilon^i_{yy} \\ \epsilon^i_{zz} \\ \epsilon^i_{xy} \\ \epsilon^i_{yz} \\ \epsilon^i_{zx} \end{pmatrix}. \tag{2.2}$$

Here the stresses of the form σ_{jj} denote normal stresses and stresses of the form σ_{jk} denote shear stresses. Similarly, ϵ_{jj} denotes normal strain and ϵ_{jk} describes shear strain. The stress and strain are related by a stiffness tensor, D, with

$$\boldsymbol{\sigma}_i = D\boldsymbol{\epsilon}_i. \tag{2.3}$$

Using the displacement in a location, the strain can be described. The displacement is given by a component in each direction, so at location i, displacement is given by

$$\mathbf{u}^i = \begin{pmatrix} u_x^i \\ u_y^i \\ u_z^i \end{pmatrix}. \tag{2.4}$$

Each component of the strain is then described using the derivative of the two corresponding components of the displacement, described by Liu, Shi, and Yang **(2020 [5])** as

$$\epsilon_{kj}^i = \frac{1}{2}\left(\frac{\partial u_j^i}{\partial k} + \frac{\partial u_k^i}{\partial j}\right), \qquad j,k \in \{x,y,z\} \tag{2.5}$$

The components of the stress vector are described using the forces in a location i, $\mathbf{p}^i$, which also consists of a component in each direction. These are described by Liu, Shi, and Yang **(2020 [5])** as

$$\frac{\partial \sigma_k^i x}{\partial x} + \frac{\partial \sigma_k^i y}{\partial y} + \frac{\partial \sigma_k^i z}{\partial z} + p_k^i = 0, \qquad k \in \{x,y,z\} \tag{2.6}$$

2.2 Finite Element Model

Using the mentioned methods to find stress and strain in a certain location, a finite element model can be established. For this the observed domain is discretized in smaller regions, within which nodes are located where the displacement, stress, strain and forces are observed. Here, a system that is used to determine the displacement from body forces will be derived for a single arbitrary element. This system in a single element can be used to built a system for the full domain using a method of global assembly. How this is exactly done depends on the exact problem that is observed.

A single element of general shape is established from a simple shape using a shape function N. This is used to map between local coordinates (ξ, η, ζ) and global coordinates (x, y, z). Using the shape function, the displacement is mapped between the local coordinate system and the global coordinate system. So the shape function is such that, for some location i,

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = N \begin{pmatrix} \xi \\ \eta \\ \zeta \end{pmatrix} \tag{2.7}$$

$$\mathbf{u}^i = N\mathbf{u}_n^i, \tag{2.8}$$

where $\mathbf{u}_n$ gives the nodal displacement in the local coordinate system. Using the derivative of the shape functions,

$$B = \begin{bmatrix} \frac{\partial N_i}{\partial x} & 0 & 0 \\ 0 & \frac{\partial N_i}{\partial y} & 0 \\ 0 & 0 & \frac{\partial N_i}{\partial z} \\ \frac{\partial N_i}{\partial y} & \frac{\partial N_i}{\partial x} & 0 \\ 0 & \frac{\partial N_i}{\partial z} & \frac{\partial N_i}{\partial y} \\ \frac{\partial N_i}{\partial z} & 0 & \frac{\partial N_i}{\partial x} \end{bmatrix},$$

a relation is established between the nodal displacement and the strain, given by

$$\boldsymbol{\epsilon}^i = B\mathbf{u}_m^i. \tag{2.9}$$

In a single element, it is required that the strain energy and the energy from body and boundary forces are equal. This is described by Borja **(2000 [1])**, for a domain Ω^e with boundary Γ^e as

$$\int_{\Omega^e} \boldsymbol{\epsilon}^T \boldsymbol{\sigma} \mathrm{d}\Omega^e - \int_{\Omega^e} \mathbf{u}^T \mathbf{f} \mathrm{d}\Omega^e + \int_{\Gamma^e} \mathbf{u}^T \mathbf{t} \mathrm{d}\Gamma^e = 0 \tag{2.10}$$

Here $\mathbf{f}$ describes the body force and $\mathbf{t}$ described the traction forces in the boundary. Now using what was established in Equation (2.3), it is obtained that

$$\int_{\Omega^e} \boldsymbol{\epsilon}^T \boldsymbol{\sigma} \mathrm{d}\Omega^e = \int_{\Omega^e} \boldsymbol{\epsilon}^T D \boldsymbol{\epsilon} \mathrm{d}\Omega^e.$$

Then using Equations (2.7) and (2.9) Equation (2.10) can be determined in nodal values as

$$\int_{\Omega^e} \mathbf{u}_n^T B^T D B \mathbf{u}_n \boldsymbol{\sigma} \mathrm{d}\Omega^e - \int_{\Omega^e} \mathbf{u}_n^T N^T \mathbf{f} \mathrm{d}\Omega^e + \int_{\Gamma^e} \mathbf{u}_n^T N^T \mathbf{t} \mathrm{d}\Gamma^e = 0$$
$$\mathbf{u}_n^T \left(\int_{\Omega^e} B^T D B \mathbf{u}_n \boldsymbol{\sigma} \mathrm{d}\Omega^e - \int_{\Omega^e} N^T \mathbf{f} \mathrm{d}\Omega^e + \int_{\Gamma^e} N^T \mathbf{t} \mathrm{d}\Gamma^e \right) = 0$$

These equations can be written as a linear system for $\mathbf{u}_n$, as

$$\begin{aligned} K \mathbf{u}_n &= \mathbf{p}_n \\ K &= \int_{\Omega^e} B^T D B \boldsymbol{\sigma}_n \mathrm{d}\Omega^e \\ \mathbf{p}_n &= \int_{\Omega^e} N^T \mathbf{f}_n \mathrm{d}\Omega^e + \int_{\Gamma^e} N^T \mathbf{t}_n \mathrm{d}\Gamma^e. \end{aligned} \tag{2.11}$$

From this system in an individual element a global system can be assembled. This is the system of the from of Equation (2.1).

Exact properties of this system differ between physical models that are being observed. However, some assumptions can be made on the matrix K. To begin with, this matrix is assumed to be non-singular and thus a solution to Equation (3.1) exists for any $\mathbf{p}^{(t)}$. Furthermore, K is assumed to be a large sparse matrix, so it has much more zero elements than non-zero elements. Lastly, K is assumed to be Symmetric Positive Definite (SPD) **(Schlumberger, 2023 [9])**.

Direct Linear Solver Methods

3

The goal of this project is to efficiently solve an linear system of the form of Equation (2.1). This chapter discusses various techniques to solve such a system, which for simplicity of notation is rewritten as

$$A\mathbf{x} = \mathbf{b}, \tag{3.1}$$

with $A \in \mathbb{R}^{n \times n}$ a matrix of coefficients, $\mathbf{x} \in \mathbb{R}^n$ a vector of unknowns and $\mathbf{b} \in \mathbb{R}^n$ right-hand side vector. In the problems observed here, as established in Chapter 2, the coefficient matrix A can be assumed to be non-singular and SPD. This means that a unique solution $\mathbf{x}$ for this system always exists.

The methods discussed in this chapter directly compute an exact solution to a linear system, without making use of iteration. The approach of the methods described here can be applied to any non-singular matrix (or any non-singular SPD matrix in the case of eigendecomposition), but these methods are not necessarily optimal for the problems observed here. As the matrices observed here are assumed to be sparse, the direct methods described in this chapter often require a lot of unnecessary computations. Furthermore, the inverse matrix of a sparse matrix is not necessarily sparse, which means potentially a lot of extra memory is required to store the inverse matrix. Although these methods may not be useful to solve the full system desired to be solved, they can be used to solve smaller systems derived in the iterative methods discussed in Chapter 4.

First the most simple way to solve a linear system, matrix inversion, is discussed in Section 3.1, together with two methods which can be used to obtain the inverse matrix. After this, an overview is given of a method, referred to as Gaussian Elimination which can be used to directly solve linear systems in Section 3.2. Lastly, an efficient substitution method to solve linear systems of triangular matrices is discussed, together with two methods to decompose a general matrix into two triangular matrices.

3.1 Matrix Inversion

The inverse of a matrix A is the matrix A^{-1}, which is such that

$$AA^{-1} = A^{-1}A = I.$$

The matrices discussed here are assumed to be non-singular, which means that an inverse always exists. Multiplying both sides of the system in Equation 3.1 from the left by A^{-1} gives a unique solution

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

Several methods can be used to compute the inverse of a matrix, two of which are discussed here. The method discussed to directly compute a solution described in Section 3.2 can also be used to compute the inverse of a matrix.

3.1.1 Inversion using the cofactor Matrix

A first direct method to compute the inverse of a matrix is by using its cofactor matrix. Using the cofactor matrix C of A, the inverse of A is obtained as

$$A^{-1} = \frac{1}{\det(A)} C^T. \tag{3.2}$$

To compute the cofactor matrix, minor matrices of A are used, where M_{ij} denotes the matrix obtained from deleting row i and column j from matrix A. Then the elements of the cofactor matrix are obtained from the determinant of these minor matrices,

$$[C]_{i,j} = (-1)^{i+j} \det(M_{ij}).$$

The computation of the inverse through this method requires $n^2 + 1$ computations of determinants, n^2 of matrices of sizes $(n-1) \times (n-1)$ and 1 of a matrix of size $n \times n$. Since the computation of a determinant can be very costly itself for large matrices, the computational cost of this method is very large. Because of this, this method is theoretically useful, but its practical use is limited.

3.1.2 Inversion using the Eigendecomposition

A second method to compute the inverse of a matrix will be derived from the eigendecomposition of the matrix. For a matrix with eigenvalues λ_i and linearly independent eigenvectors $\mathbf{v}_i$, a factorisation of the form

$$A = V \Lambda V^{-1}$$

exists. Here Λ is a diagonal matrix in which the diagonal elements are the eigenvalues of A and V is a matrix of which the columns are the eigenvectors of A. These matrices are constructed in a way such that a pair of an eigenvalue λ_i and eigenvector $\mathbf{v}_i$, which satisfy $A\mathbf{v}_i = \lambda \mathbf{v}_i$, are located in the positions of the corresponding index of their respective matrices, so if λ_i is the i'th diagonal element of Λ, then $\mathbf{v}_i$ is the i'th column of V. Since A is assumed to be positive definite, such a decomposition exists and the eigenvectors can be chosen such that they are mutually orthogonal, making V a orthogonal matrix. Furthermore, since A is non-singular, the eigenvalues of A are non-zero and thus Λ is non-singular as well.

Inversion of the eigendecomposition gives

$$(V \Lambda V^{-1})^{-1} = (V^{-1})^{-1} \Lambda^{-1} V^{-1} = V \Lambda^{-1} V^{-1}.$$

V is orthogonal, so $V^{-1} = V^T$ and thus

$$A^{-1} = V \Lambda^{-1} V^T,$$

meaning that only the inversion of Λ is required to compute the inverse of A. Since Λ is a diagonal matrix, this can be done directly, with

$$[\Lambda^{-1}]_{i,i} = \frac{1}{\lambda_i}.$$

So, if the eigendecomposition of a matrix is known, its inverse can easily be computed. However, it is still computationally costly to compute the eigendecomposition for large systems. Therefore, this method is practically not expected to be useful for the systems observed here.

3.2 Gaussian Elimination

Gaussian Elimination is a commonly used method for solving linear equation. It makes use of three types of row operations, the swapping of rows, multiplication of a row by a non-zero number and adding a multiple of one row to another row. Using these operations, a non-singular matrix can be transformed first in its row echelon form, which is an upper triangular matrix, and then in its reduced row echelon form, which is the identity matrix for a non-singular matrix **(Fraleigh and Raymond, 1995 [4])**. Applying the same transformations used to transform a matrix in its reduced echelon form transforms **b** into a unique solution **x**.

The operations of adding one row to other rows and the multiplication of rows that are required to solve a system using Gaussian elimination can be described by Gaussian transformation matrices. For the first transformation, to transform a matrix in the row echelon form, these are described by

$$\begin{aligned} M_k &= I - \boldsymbol{\alpha}^{(k)} \mathbf{e}_k^T \\ \boldsymbol{\alpha}^{(k)} &= \begin{pmatrix} 0 & \cdots & 0 & \frac{[A^{(k-1)}]_{k+1,k}}{[A^{(k-1)}]_{k,k}} & \cdots & \frac{[A^{(k-1)}]_{n,k}}{[A^{(k-1)}]_{k,k}} \end{pmatrix}^T, \end{aligned} \tag{3.3}$$

where the term $\frac{[A^{(k-1)}]_{k+1,k}}{[A^{(k-1)}]_{k,k}}$ is located in the $k+1$'th position of $\boldsymbol{\alpha}$. The matrix $A^{(k-1)}$ is the matrix obtained after multiplying the original matrix with the first $k-1$ Gaussian transformation matrices. $A^{(0)}$ is set to be the original matrix A. Using these matrices, the row echelon form can be obtained as

$$\bar{A} = M_{n-1} M_{n-2} \cdots M_1 A.$$

Since the matrices $A^{(i)}$ for $i < k$ are required to compute M_k, this calculation is usually done step-wise.

In a similar way, Gaussian transformation matrices that transform the row echelon form in the reduced row echelon can be defined by

$$\begin{aligned} N_k &= I - \mathbf{e}_k \left(\boldsymbol{\gamma}^{(k)} \right)^T \\ \boldsymbol{\gamma}^{(k)} &= \begin{pmatrix} \frac{[\bar{A}]_{1,k}}{[\bar{A}]_{k,k}} & \cdots & \frac{[\bar{A}]_{k-1,k}}{[\bar{A}]_{k,k}} & 0 & \cdots & 0 \end{pmatrix}^T, \end{aligned}$$

which is used to transform the matrix in the reduced row echelon form by

$$\tilde{A} = N_1 N_2 \cdots N_{n-1} \bar{A}.$$

The operation of swapping rows, or pivoting, is done using a pivoting matrix, which is the identity matrix with two rows swapped. So for swapping rows i and j, the matrix $P_{i,j}$ is sued, which has rows i and j swapped:

$$P_{i,j} = \begin{bmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 0 & \cdots & 1 & & \\ & & \vdots & \ddots & \vdots & & \\ & & 1 & \cdots & 0 & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{bmatrix}$$

As the diagonal element of $A^{(k-1)}$ can be zero, it might be necessary to apply a pivoting matrix before the Gaussian transformation matrix. This alters the method to obtain a row echelon form and a reduced row echelon form of a matrix to

$$\bar{A} = P_{n-1}M_{n-1}\cdots P_1M_1A$$
$$\tilde{A} = Q_1N_1\cdots Q_{n-1}N_{n-1}\bar{A},$$

where P_k and Q_k are the required pivoting matrices.

Since the reduced row echelon form is the identity matrix, it is observed that

$$Q_1N_1\cdots Q_{n-1}N_{n-1}P_{n-1}M_{n-1}\cdots P_1M_1A = I.$$

From this several useful results are obtained. First, this provides a method to obtain a solution to the system in Equation (3.1), by applying the Gaussian transformation operations to both sides of the equation. This gives

$$\begin{aligned} Q_1N_1\cdots Q_{n-1}N_{n-1}P_{n-1}M_{n-1}\cdots P_1M_1A\mathbf{x} &= Q_1N_1\cdots Q_{n-1}N_{n-1}P_{n-1}M_{n-1}\cdots P_1M_1\mathbf{b} \\ I\mathbf{x} &= Q_1N_1\cdots Q_{n-1}N_{n-1}P_{n-1}M_{n-1}\cdots P_1M_1\cdots M_1\mathbf{b} \\ \mathbf{x} &= Q_1N_1\cdots Q_{n-1}N_{n-1}P_{n-1}M_{n-1}\cdots P_1M_1\cdots M_1\mathbf{b}. \end{aligned}$$

Furthermore, the Gaussian transformation matrices can be used to describe the inverse of A. It is obtained that, apart from left multiplication of A with the Gaussian transformation matrices giving the identity matrices, right multiplication does so as well,

$$AQ_1N_1\cdots Q_{n-1}N_{n-1}P_{n-1}M_{n-1}\cdots P_1M_1 = I.$$

This means that the product of the Gaussian transformation matrices is the inverse of A, so

$$A^{-1} = Q_1N_1\cdots Q_{n-1}N_{n-1}P_{n-1}M_{n-1}\cdots P_1M_1.$$

3.3 Triangular Substitution Methods

If the matrix observed in a linear system is a triangular matrix, then the linear system can be solved in an efficient way. This is done using forward substitution for lower triangular matrices and using backward substitution for upper triangular matrices **(Vuik, C. and Lahaye, D.J.P., 2019 [12])**.

Suppose L is a non-singular lower triangular matrix and it is desired to solve linear system $L\mathbf{x} = \mathbf{b}$. Then using forward substitution, the solution of this system is obtained as

$$\begin{aligned} [\mathbf{x}]_1 &= \frac{1}{[L]_{1,1}}[\mathbf{b}]_1 \\ [\mathbf{x}]_2 &= \frac{1}{[L]_{2,2}}\left([\mathbf{b}]_2 - [L]_{2,1}[\mathbf{x}]_1\right) \\ &\vdots \\ [\mathbf{x}]_{n-1} &= \frac{1}{[L]_{n-1,n-1}}\left([\mathbf{b}]_2 - \sum_{i=1}^{n-2}[L]_{n-1,i}[\mathbf{x}]_i\right) \\ [\mathbf{x}]_n &= \frac{1}{[L]_{n,n}}\left([\mathbf{b}]_2 - \sum_{i=1}^{n-1}[L]_{n,i}[\mathbf{x}]_i\right). \end{aligned}$$

In a similar way, $U\mathbf{y} = \mathbf{c}$, with U upper triangular, can be solved using backward substitution. This is described by

$$\begin{aligned}
[\mathbf{y}]_n &= \frac{1}{[U]_{n,n}}[\mathbf{c}]_n \\
[\mathbf{y}]_{n-1} &= \frac{1}{[U]_{n-1,n-1}}\left([\mathbf{c}]_{n-1} - [U]_{n-1,n}[\mathbf{y}]_n\right) \\
&\vdots \\
[\mathbf{y}]_2 &= \frac{1}{[U]_{2,2}}\left([\mathbf{c}]_2 - \sum_{i=3}^{n}[U]_{2,i}[\mathbf{y}]_i\right) \\
[\mathbf{y}]_1 &= \frac{1}{[U]_{1,1}}\left([\mathbf{c}]_1 - \sum_{i=2}^{n}[U]_{1,i}[\mathbf{y}]_i\right).
\end{aligned}$$

Using these procedures, any linear system with a non-singular matrix A can be solved, if A is decomposed into a lower and upper triangular matrix. If lower triangular matrix L and upper triangular matrix U are such that $A = LU$, then a linear system can be written as

$$LU\mathbf{x} = \mathbf{b}.$$

This can then be solved by first solving $L\mathbf{y} = \mathbf{b}$ using forward substitution and then solving $U\mathbf{x} = \mathbf{y}$ with backward substitution. Finding triangular matrices L and U such that $LU = A$ can be done in several ways, two of which are discussed here.

A first method to decompose a matrix A in a lower and upper triangular matrix is using the LU- decomposition. This makes use of the Gaussian transformation matrices described in Section 3.2. There it was observed that a matrix A can be transformed in its row echelon form, which is upper triangular. Now it is observed that all matrices M_k defined in Equation 3.3 are all lower triangular, which means that the product and its inverse are also lower triangular. Letting U be the matrix that is obtained as the row echelon matrix, it is obtained that

$$P_{n-1}M_{n-1}\cdots P_1M_1A = U$$

This can be rewritten to

$$PA = (M_{n-1}M_{n-2}\cdots M_1)^{-1}U.$$

This means that $L = (M_{n-1}M_{n-2}\cdots M_1)^{-1}$ and $U = M_{n-1}M_{n-2}\cdots M_1A$ gives a decomposition of PA in a lower and upper triangular matrix. As LU here describes PA instead of A, the system that is solved using triangular substitution methods is

$$LU\mathbf{x} = P\mathbf{b}.$$

L can easily be obtained using the fact that

$$M_k^{-1} = I - \boldsymbol{\alpha}^{(k)}\mathbf{e}_k^T,$$

which gives

$$L = (M_{n-1}M_{n-2}\cdots M_1)^{-1} = I + \sum_{k=1}^{n-1}\boldsymbol{\alpha}^{(k)}\mathbf{e}_k^T.$$

A second way to decompose a matrix in a lower and upper matrix is using the Cholevski decomposition, which finds a matrix C such that $A = CC^T$. This decomposition uses only one matrix for both the upper and lower triangular matrix, making it advantageous in terms of used storage. However, a Cholevski decomposition is only guaranteed to exist if A is SPD. The process of finding a Cholevksi decomposition of a matrix A is described by Algorithm 3.1.

Algorithm 3.1: Cholevski Decomposition **(Vuik, C. and Lahaye, D.J.P., 2019 [12, p. 55])**

Data: Coefficient Matrix A
Result: Cholevski Decomposition Matrix C
Initialize $C = O_{n,n}$
for k=*1...n* **do**
  $[C]_{k,k} = \sqrt{[A]_{k,k} - \sum_{j=1}^{k-1} C_{k,j}^2}$
  for i=*k+1...n* **do**
    $[C]_{i,k} = \frac{1}{[C]_{k,k}} \left([A]_{i,k} - \sum_{j=1}^{k-1} C_{i,j} C_{k,j} \right)$
  end
end

Iterative Linear Solver Methods

4

Like the previous chapter, a linear system of the form

$$A\mathbf{x} = \mathbf{b} \tag{4.1}$$

is discussed, with $A \in \mathbb{R}^{n\times n}$ a large sparse matrix of coefficients, $\mathbf{x} \in \mathbb{R}^n$ a vector of unknowns and $\mathbf{b} \in \mathbb{R}^n$ right-hand side vector. Systems like this are commonly too large to apply direct methods of solving, as this requires too many computations. Iterative solvers can often be used to gain a significant advantage over direct solvers for large systems, especially if the observed matrices are sparse.

This chapter explores various iterative methods for solving sparse linear systems. The first methods discussed are basic iteration methods, such as Jacobi and Gauss-Seidel iteration, in Section 4.1. After this a transition is made to more advanced methods, such as Conjugate Gradient in Section 4.2 and GMRES in Section 4.4. The chapter closes with two different types of techniques, being multigrid methods in Section 4.5 and Domain Decomposition in Section 4.6.

4.1 Basic Iterative Methods

Most basic iterative methods make use of an iteration of the form

$$\mathbf{x}_{k+1} = G\mathbf{x}_k + \mathbf{f}. \tag{4.2}$$

Matrix $G \in \mathbb{R}^{n\times n}$ is given by

$$G = M^{-1}N = I - M^{-1}A, \tag{4.3}$$

where M and N are obtained from a splitting of A of the form $A = M - N$. This splitting of A into M and N differs between methods, but is commonly based on a splitting of A of the form

$$A = D - E - F. \tag{4.4}$$

Here $D \in \mathbb{R}^{n\times n}$ is a diagonal matrix, containing the diagonal elements of A, $E \in \mathbb{R}^{n\times n}$ is a lower triangular matrix containing the elements of A located below the diagonal, multiplied by -1 and $F \in \mathbb{R}^{n\times n}$ is a upper triangular matrix containing the elements of A above the diagonal, multiplied by -1. Vector $\mathbf{f} \in \mathbb{R}^n$ is obtained from the right-hand side vector as

$$\mathbf{f} = M^{-1}\mathbf{b}.$$

The iteration described by Equation (4.2) can be seen as solving

$$(I - G)\mathbf{x} = \mathbf{f},$$

which can be rewritten using $G = I - M^{-1}A$ to

$$M^{-1}A\mathbf{x} = M^{-1}\mathbf{b}. \tag{4.5}$$

This type of system is called a preconditioned system with M being a preconditioner.

The iteration in Equation (4.2) starts with an initial guess $\mathbf{x}_0$. There are very little requirements for this initial guess, and commonly a simple initial guess like $\mathbf{x}_0 = \mathbf{0}$ is used. The iteration stops when a predetermined stopping criterion is satisfied. This can be a chosen maximum value for k, but is more commonly based on a bound on the residual $\mathbf{r}_k = A\mathbf{x}_k - \mathbf{b}$. Then the iteration is stopped if the norm of the residual $\|\mathbf{r}_k\|$ is smaller than a chosen acceptance in residual δ.

For iterations of the from of Equation (4.2) several aspects about convergence of the iterations are known. First it is observed that the iteration satisfies

$$\mathbf{x}_{k+1} = M^{-1}N\mathbf{x}_k + M^{-1}\mathbf{b}, \tag{4.6}$$

which has limit $\mathbf{x} = M^{-1}N\mathbf{x} + M^{-1}b$. So, with $A = M - N$, this limit satisfies $A\mathbf{x} = \mathbf{b}$. Thus if Equation (4.2) converges to a solution, then that solution is a solution to the original system in Equation (4.1).

Next, it is observed that if $I - G$ is non-singular and $\mathbf{x}_*$ is an exact solution to

$$\mathbf{x} = G\mathbf{x} + \mathbf{f}, \tag{4.7}$$

then the error of the iterate obtained from Equation (4.2), $\mathbf{d}_k = \mathbf{x}_k - \mathbf{x}_*$, satisfies

$$\mathbf{d}_k = G^k\mathbf{d}_0. \tag{4.8}$$

The sequence $\mathbf{d}_k$ converges in norm if G has spectral radius ρ such that

$$\rho(G) = \max_{\lambda \in \sigma(A)} |\lambda| < 1, \tag{4.9}$$

where $\sigma(A)$ is the spectrum, the set of all eigenvalues, of A. From the fact that $\rho(G) \leq \|G\|$ for any matrix norm $\|\cdot\|$, it is then obtained that $\mathbf{x}_k - \mathbf{x}_*$ converges in norm if there is a matrix norm such that $\|G\| < 1$. Furthermore, both these conditions being satisfied leads to $I - G$ being non-singular**(Saad, 2003 [8, p.115])**. The last condition is commonly easier to use as it does not require calculating all eigenvalues of G, which is computationally costly.

Lastly, it is desired to know how fast convergence happens. The most useful way to establish the rate of convergence is by observing the general convergence factor of a method. The general convergence factor is given by

$$\phi = \lim_{k\to\infty} \left(\max_{\mathbf{x}_0 \in \mathbb{R}^n} \frac{\|\mathbf{d}_k\|}{\|\mathbf{d}_0\|} \right)^{\frac{1}{k}}. \tag{4.10}$$

For the general convergence factor it is obtained that it is equal to the spectral radius of G:

$$\begin{aligned} \phi &= \lim_{k\to\infty} \left(\max_{\mathbf{d}_0 \in \mathbb{R}^n} \frac{\|G^k\mathbf{d}_0\|}{\|\mathbf{d}_0\|} \right)^{\frac{1}{k}} \\ &= \lim_{k\to\infty} \left(\max_{\mathbf{d}_0 \in \mathbb{R}^n} \left(\|G^k\| \frac{\|\mathbf{d}_0\|}{\|\mathbf{d}_0\|} \right) \right)^{\frac{1}{k}} \\ &= \lim_{k\to\infty} \left(\|G^k\| \right)^{\frac{1}{k}} \\ &= \rho(G). \end{aligned} \tag{4.11}$$

4.1.1 Richardson's Iteration

One of the most simple iteration methods is Richardson's iteration. The iteration used for it is given by

$$\mathbf{x}_{k+1} = (I - \alpha A)\mathbf{x}_k + \alpha \mathbf{b},$$

so the iteration matrix $G_\alpha = I - \alpha A$ is used. From the combined result of Equations (4.9) and (4.11), it is observed that the method convergences if both

$$\begin{aligned} 1 - \alpha\lambda_{\min} &< 1 \\ 1 - \alpha\lambda_{\max} &> -1. \end{aligned}$$

This means that the method does not converge if A has negative eigenvalues, which it is not expected to have as A can be expected to be SPD in the problems observed here. This means that the convergence condition obtained for α is given by

$$0 < \alpha < \frac{2}{\lambda_{\max}}. \tag{4.12}$$

For this method a value of α that gives optimal convergence can also be obtained. This is obtained from minimizing the spectral radius of the iteration matrix, which is done by solving

$$-1 + \alpha\lambda_{\max} = 1 - \alpha\lambda_{\max},$$

which gives

$$\alpha_{\text{opt}} = \frac{2}{\lambda_{\max} + \lambda_{\min}}. \tag{4.13}$$

Although this value is easily derived, the actual calculation of α_{opt} can be challenging, as it requires the calculation of the eigenvalues of A.

4.1.2 Jacobi Method

The Jacobi method is another simple iteration method. It uses an iteration of the form of Equation (4.2), with $G = I - D^{-1}A$ and $\mathbf{f} = D^{-1}\mathbf{b}$, where D is the diagonal matrix obtained from a splitting of A as given in Equation (4.4). Since D is a diagonal matrix, the inverse of D can be easily calculated. The iteration obtained for the Jacobi method is as follows:

$$\mathbf{x}_{k+1} = D^{-1}(E + F)\mathbf{x}_k + D^{-1}\mathbf{b} \tag{4.14}$$

The Jacobi iteration can also be viewed as solving the preconditioned system given in Equation (4.5) with precondition matrix $M = D$.

For this iteration method a damped or weighted variant exists, that is based on a weighted average of the current iterant, $\mathbf{x}_k$ and the next iterant obtained from applying a step of Jacobi iteration. This gives

$$\mathbf{x}_{k+1} = (1 - \omega)\mathbf{x}_k + \omega(D^{-1}(E + F)\mathbf{x}_k + D^{-1}\mathbf{b}). \tag{4.15}$$

This corresponds to an iteration of the form of Equation (4.2) with $G = I - \omega D^{-1}A$. This damped Jacobi method also corresponds to applying the Richardson method with $\alpha = \frac{1}{\omega}$ to $D^{-1}A = D^{-1}\mathbf{b}$ **(Vuik, C. and Lahaye, D.J.P., 2019 [12])**.

4.1.3 Gauss-Seidel Method

Another simple iteration method is the Gauss-Seidel iteration. Like Jacobi iteration this uses an iteration of the form of Equation (4.2), but for this method $G = I - (D - E)^{-1}A$ is used. This gives the iteration

$$\mathbf{x}_{k+1} = (D - E)^{-1}F\mathbf{x}_k + (D - E)^{-1}\mathbf{b}. \tag{4.16}$$

Unlike the iteration for the Jacobi method, which only has the easily computed inverse D^{-1}, this iteration uses a more complicated inverse $(D - E)^{-1}$. Since $D - E$ is a lower triangular matrix, there is in general no easy way to compute this inverse. Therefore the iteration is instead done by solving the system

$$(D - E)\mathbf{x}_{k+1} = F\mathbf{x}_k + \mathbf{b}.$$

This system can be solved directly using forward substitution. The Gauss-Seidel method can also be described by the preconditioned system in Equation (4.5) with precondition matrix $M = D - E$.

Similarly to the Gauss-Seidel method, a backward Gauss-Seidel method is defined. In each step of the backward Gauss-Seidel iteration

$$(D - F)\mathbf{x}_{k+1} = E\mathbf{x}_k + \mathbf{b}.$$

is solved. This is the same as the preconditioned system using $M = D - F$. Forward and backward Gauss-Seidel can be used together to obtain the Symmetric Gauss-Seidel method. This uses an iteration consisting of a forward Gauss-Seidel iteration followed by a backward Gauss-Seidel iteration.

4.1.4 SOR and SSOR Method

Both the Jacobi and Gauss-Seidel methods use a splitting of A of the form $A = M - N$, with $M = D$ for Jacobi and $M = D - E$ for Gauss-Seidel. The splitting is always done in the same way for these methods. Overrelaxation uses a splitting which can differ depending on a parameter ω given by

$$\omega A = (D - \omega E) - (\omega F + (1 - \omega)D).$$

This gives rise to Successive Over Relaxation(SOR) method, which uses iteration

$$(D - \omega E)\mathbf{x}_{k+1} = (\omega F + (1 - \omega)D)\mathbf{x}_k + \omega\mathbf{b}. \tag{4.17}$$

This corresponds to the preconditioned system with $M = \frac{1}{\omega}(D - \omega E)$.

Like Gauss-Seidel, a backward SOR iteration is obtained from the exchange of E and F in the iteration step. From combining the forward and backward SOR iterations a Symmetric SOR method is obtained. Similar to symmetric Gauss-Seidel, this method consists of first performing a forward iteration and following it up with a backward iteration. This gives the two iterations

$$\begin{aligned}(D - \omega E)\mathbf{x}_{k+\frac{1}{2}} &= (\omega F + (1 - \omega)D)\mathbf{x}_k + \omega\mathbf{b}\\ (D - \omega F)\mathbf{x}_{k+1} &= (\omega E + (1 - \omega)D)\mathbf{x}_{k+\frac{1}{2}} + \omega\mathbf{b}.\end{aligned}$$

These two iterations can be written as one iteration of the form of Equation (4.2) with

$$G_\omega = (D - \omega F)^{-1}(-I + (2 - \omega)D(D - \omega E)^{-1}(\omega F + (1 - \omega)D) \tag{4.18}$$

$$\mathbf{f}_\omega = \omega(2 - \omega)(D - \omega F)^{-1}D(D - \omega E)^{-1}\mathbf{b} \tag{4.19}$$

As the preconditioned system from Equation (4.5) this method uses $M = \frac{1}{\omega(2-\omega)}(D - \omega E)D^{-1}(D - \omega F)$.

For both the SOR and the SSOR method, the performance highly depends on the choice of parameter ω. For this choice, it is first noted that ω is required to be in $(0, 2)$, as any value outside of this range causes the method to be unstable. $\omega \in (0, 2)$ ensures convergence of the iterations. A commonly used choice for SSOR is the simple choice of $\omega = 1$. This simple choice turns SOR into the Gauss-Seidel iteration. A found optimal choice of ω is given by

$$\omega_{\text{opt}} = \frac{2}{1 + \sqrt{1 - \rho(G_J)}}.$$

Here G_J is the iteration matrix of the Jacobi method, given in Equation (4.14) **(Demmel, 1997 [2])**. Using this optimal value for ω does require calculation of the eigenvalues of the iteration matrix of the Jacobi method, which can be computationally costly.

4.2 Conjugate Gradient Method

All the methods seen in the previous section use a simple iteration, which is the same at each step. In general these methods work well, but the convergence of these methods can slow down significantly if the spectral radius of the iteration matrix is close to 1. In problems obtained from discretised PDE problems the spectral radius commonly gets closer to 1 if the size of the problem increases, which means that for large systems, simple iteration methods might not be sufficient.

Improvements in simple iteration techniques are often derived from techniques based on projections onto a Krylov subspace. These techniques try to approximate $A^{-1}\mathbf{b}$ by $p(A)\,\mathbf{b}$, where $p(A)$ is a polynomial which has a degree depending on the degree of the Krylov subspaces. A method of this type that works particularly well for SPD matrices is the Conjugate Gradient (CG) method.

In the Conjugate Gradient method a projection is performed on the Krylov subspace $\mathcal{K}_m(\mathbf{r}_0, A)$ given by

$$\mathcal{K}_m(\mathbf{r}_0, A) = \text{Span}\left\{\mathbf{r}_0, A\mathbf{r}_0, A^2\mathbf{r}_0, \ldots A^{m-1}\mathbf{r}_0\right\} \tag{4.20}$$

$\mathbf{r}_0$ is the residual of the initial guess $\mathbf{x}_0$, so $\mathbf{r}_0 = A\mathbf{x}_0 - \mathbf{b}$. This produces a solution $\mathbf{x}_m \in \mathbf{x}_0 + \mathcal{K}_m$ that has a residual orthogonal to the Krylov subspace $\mathcal{K}_m$.

4.2.1 Conjugate Gradient Algorithm

The method is derived from the Lanczos algorithm, which finds a solution $\mathbf{x}_m \in \mathbb{R}^n$ from

$$\begin{aligned} \mathbf{x}_m &= \mathbf{x}_0 + V_m\mathbf{y}_m \\ \mathbf{y}_m &= T_m^{-1}(\beta\mathbf{e}_1). \end{aligned} \tag{4.21}$$

The matrix $V_m \in \mathbb{R}^{n\times m}$ is obtained from the Lanczos vectors $\mathbf{v}_j \in \mathbb{R}^n$, which are orthogonal vectors that span the Krylov subspace $\mathcal{K}_m(\mathbf{r}_0, A)$. These vectors are obtained by the Lanczos method, which is an orthonogalisation procedure that works particularly well when producing a Krylov subspace for an SPD matrix A. Matrix T_m is a tridiagonal matrix built from the orthogonalisation coefficients derived in this same procedure. Saad **(2003 [8, p. 196])** describes the full Lanczos method and provides an algorithm for the calculation of the orthogonal vectors and the coefficients used to built the matrices V_m and T_m. The scalar β is obtained from the norm of the initial residual, $\beta = \|\mathbf{r}_0\|_2$. $\mathbf{e}_1 \in \mathbb{R}^m$ denotes the first unit vector of length m. This

method allows for the direct computation of the residual obtained after applying the method using only what was obtained at the last step of the algorithm by

$$\mathbf{r}_m = \beta_{m+1}\mathbf{e}_m^T\mathbf{y}_m\mathbf{v}_{m+1}. \tag{4.22}$$

Here $\beta_{m+1}, \mathbf{y}_m$ and $\mathbf{v}_{m+1}$ are all computed at the last step of the Lanczos procedure and $\mathbf{e}_m$ denotes the m'th unit vector of length m. Here it is seen that $\mathbf{r}_m = \sigma_m\mathbf{v}_{m+1}$, where $\sigma_m = \beta_{m+1}\mathbf{e}_m^T\mathbf{y}_m$ is a scalar. Then since the vectors $\mathbf{v}_i$ are orthogonal to each other, the residuals are orthogonal to each other as well.

To describe the Conjugate Gradient method another set of vectors, $\mathbf{p}_j$, is required, which form a A-conjugate set, $(A\mathbf{p}_i, \mathbf{p}_j) = 0$ for $i \neq j$. These vectors are obtained as the columns from the matrix $P_m = V_m U_m^{-1}$, where U_m is obtained from the LU-decomposition of T_m. With these vectors the iteration for the CG method is derived as

$$\mathbf{x}_{j+1} = \mathbf{x}_j + \alpha_j\mathbf{p}_j, \tag{4.23}$$

where $\alpha_j = \frac{(\mathbf{r}_j,\mathbf{r}_j)}{(A\mathbf{p}_j,\mathbf{p}_j)}$. From this recurrence a recurrence for the residuals is obtained as

$$\mathbf{r}_{j+1} = \mathbf{r}_j - \alpha_j A\mathbf{p}_j. \tag{4.24}$$

Now, computing the vectors $\mathbf{p}_j$ from the matrix P_m is computationally very inefficient, as this would require a matrix product of V_m and U_M, which both require performing the Lanczos procedure. Fortunately, a recurrence relation for $\mathbf{p}_j$ can be obtained as

$$\mathbf{p}_{j+1} = \mathbf{r}_{j+1} + \beta_j\mathbf{p}_j, \tag{4.25}$$

with $\beta_j = \frac{(\mathbf{r}_{j+1},\mathbf{r}_{j+1})}{(\mathbf{r}_j,\mathbf{r}_j)}$. This means that all required vectors at each step can be described by data obtained in the same or previous step.

Algorithm 4.1 describes the Conjugate Gradient method. The convergence condition used is usually in the form of a upper bound on the residual, $\|\mathbf{r}_j\| < a$, $a \in \mathbb{R}$. Only one matrix-vector product is required, as $A\mathbf{p}_j$ can be stored after the computing it the first time it is required. Because only the previously calculated variants of vectors and scalars are required, only four vectors are required to be stored.

4.2.2 Estimate of eigenvalues of coefficient matrix

An advantage of the Lanczos algorithm is directly obtaining the tridiagonal matrix $T_m \in \mathbb{R}^{m\times m}$, of which the largest and smallest eigenvalues give a good estimate of the largest and smallest eigenvalue of A. Here m denotes the amount of steps required in the CG algorithm to satisfy the convergence condition. From these eigenvalues, an estimate of the condition number of A can be obtained, which gives valuable information about the convergence of iteration methods like the CG and Lanczos algorithm. The computation of eigenvalues of a tridiagonal matrix is much easier than that of a general matrix A.

In the Conjugate Gradient method such a matrix is not computed directly, but the same tridiagonal matrix T_m can also be computed using the information computed in the Conjugate Gradient method. T_m is of the form

$$T_m = \text{tridiag}[\eta_j, \delta_j, \eta_{j+1}],$$

Algorithm 4.1: Conjugate Gradient Method **(Saad, 2003 [8, p. 199])**

Data: Coefficient Matrix A, Right-hand side Vector $\mathbf{b}$, Initial Guess $\mathbf{x}_0$
Result: Solution $\mathbf{x}$, Residual $\mathbf{r}$
Initialize $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$
$\mathbf{p}_0 = \mathbf{r}_0$
$j = 0$
while *convergence condition not satisfied* **do**
  $\mathbf{q}_j = A\mathbf{p}_j$
  $\alpha_j = \frac{(\mathbf{r}_j, \mathbf{r}_j)}{(\mathbf{q}_j, \mathbf{p}_j)}$
  $\mathbf{x}_{j+1} = \mathbf{x}_j + \alpha_j \mathbf{p}_j$
  $\mathbf{r}_{j+1} = \mathbf{r}_j - \alpha_j \mathbf{q}_j$
  $\beta_j = \frac{(\mathbf{r}_{j+1}, \mathbf{r}_{j+1})}{(\mathbf{r}_j, \mathbf{r}_j)}$
  $\mathbf{p}_{j+1} = \mathbf{r}_{j+1} + \beta_j \mathbf{p}_j$
  $j += 1$
end
$\mathbf{x} = \mathbf{x}_j$
$\mathbf{r} = \mathbf{r}_j$

where η_j and δ_j are obtained directly from the coefficients calculated in the CG algorithm. These coefficients are obtained from

$$\delta_{j+1} = \begin{cases} \frac{1}{\alpha_j} & , j = 0 \\ \frac{1}{\alpha_j} + \frac{\beta_{j-1}}{\alpha_{j-1}} & , j \neq 0 \end{cases}$$
$$\eta_{j+1} = \frac{\sqrt{\beta_{j-1}}}{\alpha_{j-1}}$$

Note here how for η_j no special case for $j = 0$ is required, as the first η_j that is used is η_1. The calculation of these coefficients can either be done during each step of the CG algorithm or afterwards. Neither is problematic regarding computation or storage. The only added computation is only a calculation on scalars. The extra storage required consists of two or four vectors of length m, depending on whether the values of α and β are stored.

4.2.3 Convergence behaviour of the CG method

With the derived method for finding an estimate of the solution of the system in Equation (4.1) with the Conjugate Gradient method, it is desired to know how good this estimate is and what the convergence behaviour of this estimate is. After m steps of the CG method, an approximate solution $\mathbf{x}_m$ is obtained. This $\mathbf{x}_m$ minimises the $A-$norm of the error, $\mathbf{d}_m = \mathbf{x}_* - \mathbf{x}_m$, with $\mathbf{x}_*$ the exact solution of Equation (4.1), in the subspace $\mathbf{x}_0 + \mathcal{K}_m$. Since $\mathcal{K}_m$ consists of the set of all vectors of the form $\mathbf{x}_0 + p_m(A)\,\mathbf{r}_0$ with p_m a polynomial of degree $\leq m-1$, the obtained approximate solution is such that $\mathbf{x} = \mathbf{x}_0 + p_m(A)\,\mathbf{r}_0$. Since $\mathbf{x}$ is the minimizes the $A-$norm of the error, the polynomial p_m is such that

$$\|(I - Ap_m(A))\,\mathbf{d}_0\|_A = \min_{q \in \mathbb{P}_{m-1}} \|(I - Ap(A))\,\mathbf{d}_0\|_A, \tag{4.26}$$

so p_m gives the best approximate solution $\mathbf{x}$ in $\mathbf{x}_0 + \mathcal{K}_m$. From this it can easily be concluded that the approximate solutions of the CG method converges to the exact solution, as $\mathbf{x}_0 + \mathcal{K}_m \subseteq$

$\mathbf{x}_0 + \mathcal{K}_{m+1}$ and

$$\min_{q \in \mathbb{P}_m} \| \left(I - Ap\left(A\right)\right) \mathbf{d}_0 \|_A \leq \min_{q \in \mathbb{P}_m} \| \left(I - Ap\left(A\right)\right) \mathbf{d}_0 \|_A.$$

Furthermore, a formula for the rate of convergence of the CG method is known, given by

$$\|\mathbf{x}_* - \mathbf{x}_m\|_A \leq 2 \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^m \|\mathbf{x}_* - \mathbf{x}_0\|_A.$$

κ denotes the condition number of A, which may be challenging to calculate. However, a good estimate of the condition number can be obtained, as described in Section 4.2.2. From this relation an expected amount of iterations can be obtained for a desired maximum tolerance of error ϵ, which is such that $\|\mathbf{x}_* - \mathbf{x}\|_A < \epsilon$, with $\mathbf{x}$ the solution obtained from the CG method. This would require

$$2 \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^m \|\mathbf{x}_* - \mathbf{x}_0\|_A < \epsilon,$$

so

$$m > \log_{\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}} \left(\frac{\epsilon}{\|\mathbf{x}_* - \mathbf{x}_0\|_A} \right) = \frac{\log\left(\frac{\epsilon}{\|\mathbf{x}_* - \mathbf{x}_0\|_A}\right)}{\log\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)}$$

4.2.4 Variations of the Conjugate Gradient method

Aside from the default formulation of the Conjugate Gradient method, several methods exist that are similar to this method, but satisfy different requirements. Two methods similar to CG are described here. First the Conjugate Residual method, which is very similar to CG, is described shortly. Then the BiCGSTAB method is described more extensively in Section 4.3. Other variations exist, but will not be discussed extensively as they are variations of the two methods that are discussed and are expected to perform worse than these two methods. These include the Generalised CR and ORTHOMIN methods, which are similar to the CR method, and the BiCG method and CG Stabilized method, which are used to derive the BiCGSTAB method.

The Conjugate Residual method requires the residual to be A-orthogonal and the vectors $\mathbf{q}_j = A\mathbf{p}_j$ to be orthogonal. The algorithm for this method is very similar, but because of the requirement of the residual being A-orthogonal instead of orthogonal and $\mathbf{q}_j$ being orthogonal instead of A-orthogonal, the scalars α and β are calculated differently then in the CG method

$$\alpha_j = \frac{(\mathbf{r}_j, A\mathbf{r}_j)}{(\mathbf{q}_j, \mathbf{q}_j)}$$
$$\beta_j = \frac{(\mathbf{r}_{j+1}, A\mathbf{r}_{j+1})}{(\mathbf{r}_j, A\mathbf{r}_j)}.$$

To satisfy the requirement of $\mathbf{q}_j$ being orthogonal, it now is calculated by

$$\mathbf{q}_{j+1} = A\mathbf{r}_{j+1} + \beta_j \mathbf{q}_j.$$

In general the original CG method is preferred over this method, as the Conjugate Residual method requires extra storage of the vector $A\mathbf{r}_j$ and an extra computation in the calculation of $\mathbf{q}_j$ when compared to the CG method. In some cases this method however turns out to be useful.

4.3 BiCGSTAB

The Conjugate Gradient method is designed specifically for systems with SPD matrices, which means that the convergence of the method is most robust when applied to systems with SPD matrices. If CG is applied to a general matrix A that is not necessarily SPD, the convergence is expected to be slower and less robust. BiCGSTAB or Biconjugate Gradient Stabilised makes use of a similar iteration as CG, but its convergence can be expected to not depend on whether the coefficient matrix is SPD. BiCGSTAB makes use of projection on the Krylov subspace

$$\mathcal{K}_m\left(\mathbf{v}_1, A\right) = \text{span}\left(\mathbf{v}_1, A\mathbf{v}_1, A^2\mathbf{v}_1, \ldots A^{m-1}\mathbf{v}_1\right)$$

and orthogonal to the subspace

$$\mathcal{L}_m\left(\mathbf{w}_1, A\right) = \text{span}\left(\mathbf{w}_1, A\mathbf{w}_1, A^2\mathbf{w}_1, \ldots A^{m-1}\mathbf{w}_1\right),$$

where $\mathbf{v} = \frac{\mathbf{r}_0}{\|\mathbf{r}_0\|}$ and $\mathbf{w}$ is arbitrary, but not orthogonal to $\mathbf{v}$. The method uses, apart from the residual of the system that is desired to be solved, the residual of a dual system $A^T x^* = b^*$. Since in the cases observed here A is symmetric, this system is the same as the original system, and thus the exact solution will be the same as well. However, the residual of the dual system, $\mathbf{r}_j^*$ is at each step obtained from the arbitrarily initialised vector $\mathbf{w}_1$ and thus is different from the residual of the original system $\mathbf{r}_j$. At each step of the BiCGSTAB algorithm both these residuals are used in the calculation of the coefficients α_j and β_j, which, like in the original CG algorithm, are used to determine the residuals at the next step. These coefficients are calculated such that residuals original system are orthogonal to the residuals of the dual system; $\left(\mathbf{r}_i, \mathbf{r}_j^*\right) = 0, i \neq j$.

Residuals and vectors $\mathbf{p}_j$ are obtained from the relations

$$\begin{aligned}\mathbf{r}_j &= \psi_j\left(A\right)\phi_j\left(A\right)\mathbf{r}_0\\ \mathbf{p}_j &= \psi_j\left(A\right)\pi_j\left(A\right)\mathbf{p}_0,\end{aligned}$$

with

$$\begin{aligned}\phi_{j+1}\left(t\right) &= \phi_j\left(t\right) - \alpha_j t\pi_j\left(t\right)\\ \pi_{j+1}\left(t\right) &= \phi_{j+1}\left(t\right) - \beta_j t\pi_j\left(t\right)\\ \psi_{j+1}\left(t\right) &= \left(1-\omega_j\right)\psi_j\left(t\right).\end{aligned}$$

The scalars α_j, β_j and ω_j are obtained as

$$\begin{aligned}\alpha_j &= \frac{\left(\psi_j\left(A\right)\phi_j\left(A\right)\mathbf{r}_0, \mathbf{r}_0^*\right)}{\left(A\psi_j\left(A\right)\pi_j\left(A\right)\mathbf{r}_0, \mathbf{r}_0^*\right)}\\ \omega_j &= \frac{\left(A\mathbf{s}_j, \mathbf{s}_j\right)}{\left(A\mathbf{s}_j, A\mathbf{s}_j\right)}\\ \beta_j &= \frac{\alpha_j\left(\phi_{j+1}\left(A\right)\mathbf{r}_0, \psi_{j+1}\left(A^T\right)\mathbf{r}_0^*\right)}{\omega_j\left(\phi_j\left(A\right)\mathbf{r}_0, \psi_j\left(A^T\right)\mathbf{r}_0^*\right)},\end{aligned}$$

where $\mathbf{s}_j = \mathbf{r}_j - \alpha_j A\mathbf{p}_j$. Obviously it is desired to use a recurrence relation instead of computing the polynomials of A, which gives rise to the BiCGSTAB algorithm in Algorithm 4.2

An advantage of BiCGSTAB is that it gains faster convergence than the regular CG method and than BiCG. The CGS method does this as well, but requires squaring the residual polynomials, which can cause substantial build-up of errors. Another advantage of BiCG over similar variations of CG like BiCG is that it does not require the use of A^T, although this does not matter if A is symmetric.

Algorithm 4.2: BiCGSTAB Method **(Saad, 2003 [8, p. 246])**

Data: Coefficient Matrix A, Right-hand side Vector $\mathbf{b}$, Initial Guess $\mathbf{x}_0$, Arbitrarily Chosen $\mathbf{r}_0^*$.
Result: Solution $\mathbf{x}$, Residual $\mathbf{r}$
Initialize $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$
$\mathbf{p}_0 = \mathbf{r}_0$
$j = 0$
while *convergence condition not satisfied* **do**
$\mathbf{q}_j = A\mathbf{p}_j$
$\alpha_j = \frac{(\mathbf{r}_j, \mathbf{r}_0^*)}{(\mathbf{q}_j, \mathbf{r}_0^*)}$
$\mathbf{s}_j = \mathbf{r}_j - \alpha_j \mathbf{q}_j$
$\mathbf{t}_j = A\mathbf{s}_j$ $\omega = \frac{(\mathbf{t}_j, \mathbf{s}_j)}{(\mathbf{t}_j, \mathbf{t}_j)}$ $\mathbf{x}_{j+1} = \mathbf{x}_j + \alpha_j \mathbf{p}_j + \omega_j \mathbf{s}_j$
$\mathbf{r}_{j+1} = \mathbf{s}_j - \omega_j \mathbf{t}_j$
$\beta_j = \frac{(\mathbf{r}_{j+1}, \mathbf{r}_0^*)}{(\mathbf{r}_j, \mathbf{r}_j)}$
$\mathbf{p}_{j+1} = \mathbf{r}_{j+1} + \beta\,(\mathbf{p}_j - \omega_j A\mathbf{p}_j)$
$j += 1$
end
$\mathbf{x} = \mathbf{x}_j$
$\mathbf{r} = \mathbf{r}_j$

4.4 GMRES

Like the Conjugate Gradient method, the GMRES method tries to approximate $A^{-1}\mathbf{b}$ by $p(A)\,\mathbf{b}$, with $p(A)$ a polynomial corresponding to a projection on a Krylov subspace of A. The Krylov subspace used for GMRES is slightly different from the one used in the CG method and is given by $\mathcal{K}_m(\mathbf{v}_1, A)$, where $\mathbf{v}_1 = \frac{1}{\|\mathbf{r}_0\|}\mathbf{r}_0$, constructed in the same way as the Krylov subspace in Equation (4.20). Since the only difference is $\mathbf{v}_1$ being a scaled variant of $\mathbf{r}_0$, projection on this Krylov subspace also minimises the residual $\mathbf{r}_m$ for $\mathbf{x}_m \in \mathbf{x}_0 + \mathcal{K}_m$. The difference between the two methods lies in the fact that the residual in the GMRES is such that it is orthogonal to the subspace

$$\mathcal{L}_m(\mathbf{v}_1, A) = A\mathcal{K}_m(\mathbf{v}_1, A) = \text{Span}\left\{A\mathbf{v}_1, A^2\mathbf{v}_1, A^3\mathbf{v}_1, \dots A^m\mathbf{v}_1\right\},$$

where the residual was orthogonal to $\mathcal{K}_m$ in the CG method.

4.4.1 The GMRES Algorithm

The process of finding a solution using the GMRES method is based on the fact that any vector $\mathbf{x}_m \in \mathbf{x} + \mathcal{K}_m$ can be expressed as

$$\mathbf{x} = \mathbf{x}_0 + V_m \mathbf{y}_m. \tag{4.27}$$

Matrix V_m is similar as it was for the CG method in Equation (4.21), as its columns form an orthogonal basis for the Krylov subspace $\mathcal{K}$, but it is constructed in a different way. The residual $\mathbf{r}_m$ is now obtained as

$$\mathbf{r}_m = \mathbf{r}_0 - AV_m \mathbf{y}_m. \tag{4.28}$$

Now let $\beta = \frac{1}{\|\mathbf{r}_0\|}$, so $\mathbf{r}_0 = \beta\mathbf{v}_1$, then since $\mathbf{v}_1$ is the first column of V_k, for any k. This means that $\mathbf{r}_0 = \beta V_k \mathbf{e}_1$, where $\mathbf{e}_1$ is the first unity vector. Furthermore, the product AV_m can be rewritten

as $AV_m = V_{m+1}\bar{H}_m$, where $\bar{H}_m \in \mathbb{R}^{(m+1)\times m}$ is the so called Hessenberg matrix obtained from deleting the last row of $H_m = V_m^T A V_m$. Arnoldi's algorithm **(Saad, 2003 [8, p. 160])** gives an efficient method to compute the entries of H_m. From this it is concluded that the residual can be obtained from

$$\mathbf{r}_m = V_{m+1}\left(\beta\mathbf{e}_1 - \bar{H}_m\mathbf{y}_m\right). \tag{4.29}$$

It is desired to minimize this residual in norm. Now, it is obtained that

$$\begin{aligned}\|\mathbf{r}_m\| &= \|V_{m+1}\left(\beta\mathbf{e}_1 - \bar{H}_m\mathbf{y}_m\right)\| \\ &= \|V_{m+1}\|\|\beta\mathbf{e}_1 - \bar{H}_m\mathbf{y}_m\| \\ &= \|\beta\mathbf{e}_1 - \bar{H}_m\mathbf{y}_m\|,\end{aligned}$$

as V_{m+1} is orthonormal. Thus the minimal residual is obtained from minimising $\beta\mathbf{e}_1 - \bar{H}_m\mathbf{y}_m$. Minimising this vector is computationally inexpensive compared to computations on the original system, since $\bar{H}_m$ is small compared to the original coefficient matrix A.

The biggest challenge in the GMRES algorithm is obtaining the matrices V_{m+1} and H_m efficiently. For this two methods are commonly used, both based on different variants of the Arnoldi process. The first function is a basic version using Modified Gram-Schmidt orthogonalisation, the second version uses a Householder variant for orthogonalisation. Since the Householder variant is numerically more robust, only this variant will be discussed extensively here. [8] goes over both methods more extensively.

For the Householder variant of the Arnoldi method a series of makes use of the Householder reflection matrices $P_k \in \mathbb{R}^{n\times n}$, given by

$$P_k = I - 2\mathbf{w}_k\mathbf{w}_k^T, \tag{4.30}$$

where $\mathbf{w}_k$ is constructed from the elements of A. With these Householder matrices the columns of V_m can be obtained as

$$\mathbf{v}_j = P_1 P_2 \cdots P_j e_j. \tag{4.31}$$

This means that $\mathbf{x}_m$ can be obtained as

$$\mathbf{x}_m = \mathbf{x}_0 + P_1([\mathbf{y}_m]_1 e_1 + P_2([\mathbf{y}_m]_2 e_2 + \cdots P_{m-1}([\mathbf{y}_m]_{m-1}e_{m-1} + P_m[\mathbf{y}_m]_m e_m))). \tag{4.32}$$

The computational requirement of this iterative computation of $\mathbf{x}_m$ is negligible compared to the orthogonalisation process, as it takes roughly the same amount of operations as the calculation of the last Arnoldi vector $\mathbf{v}_m$. From this the Householder GMRES Algorithm given in Algorithm 4.3 is obtained.

4.4.2 Minimization method

To compute $\mathbf{y}_m$ a minimization method is required. An elegant technique to solve this problem is by making use of plane rotations to transform matrix $\bar{H}_m$ into an upper triangular matrix. The plane rotation is done by multiplying $\bar{H}_m$ by a rotation matrices of the form

$$\Omega_i = \begin{bmatrix} 1 & & & & & & & \\ & \ddots & & & & & & \\ & & 1 & & & & & \\ & & & c_i & s_i & & & \\ & & & -s_i & c_i & & & \\ & & & & & 1 & & \\ & & & & & & \ddots & \\ & & & & & & & 1 \end{bmatrix}. \tag{4.33}$$

Algorithm 4.3: Householder GMRES Method **(Saad, 2003 [8, p. 174])**

Data: Coefficient Matrix A, Right-hand side Vector $\mathbf{b}$, Initial Guess $\mathbf{x}_0$.
Result: Solution $\mathbf{x}$
Initialize $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$
$\mathbf{z} = \mathbf{r}_0$
for *j = 1, ... m+1* **do**
  Compute unit vector $\mathbf{w}_j$ such that
  - $[\mathbf{w}_j]_i = 0, i = 1, \dots j-1$
  - $[\mathbf{z} - 2\mathbf{w}_j\mathbf{w}_j^T\mathbf{z}]_i = 0, i = j+1 \dots n$

  $P_j = I - 2\mathbf{w}_j\mathbf{w}_j^T$
  $\mathbf{h}_{j-1} = \mathbf{z} - 2\mathbf{w}^T\mathbf{z}\mathbf{w}$
  $\mathbf{v} = P_1P_2\cdots P_j\mathbf{e}_j$
  $\mathbf{z} = P_jP_{j-1}\cdots P_1A\mathbf{v}$
  if $[\mathbf{h}_{j-1}]_j = 0$ **then**
    Stop
  end
end
Define $\beta = \mathbf{e}_1^T\mathbf{h}_0 = [\mathbf{h}_0]_1$
Define $\bar{H}_m = [\mathbf{h}_1, \mathbf{h}_m]_{[1:m+1,1:m]}$
Compute $\mathbf{y}_m = \text{Argmin}_{\mathbf{y}}\|\beta\mathbf{e}_1 - \bar{H}_m\mathbf{y}\|$ using minimization method
for $m, m-1, \dots 1$ **do**
  $\mathbf{z} = P_j([\mathbf{y}_m]_j\mathbf{e}_j + \mathbf{z})$
end
$\mathbf{x} = \mathbf{x}_0 + \mathbf{z}$

To use this plane rotation, we define $\bar{\mathbf{g}}_0 = \beta\mathbf{e}_1$, which is the right hand side in the least squares problem that is desired to be solved, and $\bar{H}_m^{(0)} = \bar{H}_m$. These are updated, by

$$\bar{H}_m^{(i)} = \Omega_i\bar{H}_m^{(i-1)} \tag{4.34}$$
$$\bar{\mathbf{g}}_i = \Omega_i\bar{\mathbf{g}}_i, \tag{4.35}$$

with Ω_i having

$$s_1 = \frac{h_{21}}{\sqrt{h_{11}^2 + h_{21}^2}} \tag{4.36}$$
$$s_i = \frac{h_{i+1,i}}{\sqrt{\left(h_{ii}^{(i-1)}\right)^2 + h_{i+1,i}^2}} \tag{4.37}$$
$$c_1 = \frac{h_{11}}{\sqrt{h_{11}^2 + h_{21}^2}} \tag{4.38}$$
$$c_i = \frac{h_{ii}^{(i-1)}}{\sqrt{\left(h_{ii}^{(i-1)}\right)^2 + h_{i+1,i}^2}}. \tag{4.39}$$

Although these updates require a matrix-matrix multiplication and a matrix-vector multiplication, they are computationally not problematic. This is because the size of $\bar{H}_m$ is in general

small compared to the size of the vectors in the **for**-loop in Algorithm 4.3. Furthermore, updates are only required in two rows of $\bar{H}_m$ and two elements of $\bar{\mathbf{g}}_i$, because of the way in which Ω_i is built. For this update it is seen that, since Ω_i is unitary,

$$\begin{aligned}\|\mathbf{g}_i - \bar{H}_m^{(i)} y\| &= \|\Omega_i\| \|\mathbf{g}_i - \bar{H}_m^{(i)} y\| \\ &= \|\Omega_i \mathbf{g}_i - \Omega_i \bar{H}_m^{(i)} y\| \\ &= \|\mathbf{g}_{i+1} - \Omega_i \bar{H}_m^{(i+1)} y\|.\end{aligned}$$

This means that

$$\text{Argmin}_{\mathbf{y}} \|\bar{\mathbf{g}}_0 - \bar{H}_m \mathbf{y}\| = \text{Argmin}_{\mathbf{y}} \|\bar{\mathbf{g}}_m - \bar{H}_m^{(m)} \mathbf{y}\|. \tag{4.40}$$

Doing these updates m times provides $\bar{\mathbf{g}}_m \in \mathbb{R}^{m+1}$ and matrix $\bar{H}_m^{(m)} \in \mathbb{R}^{m+1 \times m}$, which consists of an upper triangular $R_m \in \mathbb{R}^{m \times m}$ and a row of zeroes. From these a square system can be easily obtained, by taking the upper triangular matrix R_m and defining g_m as the first m elements of $\bar{\mathbf{g}}_m$. Then since the last row of $\bar{H}_m^{(m)}$ consists of only zeroes,

$$\text{Argmin}_{\mathbf{y}} \|\bar{\mathbf{g}}_m - \bar{H}_m^{(m)} \mathbf{y}\| = \text{Argmin}_{\mathbf{y}} \|\mathbf{g}_m - R_m \mathbf{y}\|. \tag{4.41}$$

Now, as stated by Saad **(2003 [8, p. 176])**, R_m is non-singular if A is non-singular. This is the case here and thus

$$\mathbf{y}_m = R_m^{-1} \mathbf{g}_m \tag{4.42}$$

$$\mathbf{r}_m = |[\bar{\mathbf{g}}_m]_{m+1}|. \tag{4.43}$$

4.4.3 Stopping Condition

The GMRES method as discussed in Section 4.4.1 is usually applied with a chosen number of iterations, m. In Algorithm 4.3 a stopping condition is added if $[\mathbf{h}_{j-1}]_j = 0$, which is a result of the method to compute the least squares solution $\|\beta \mathbf{e}_1 - \bar{H}_m \mathbf{y}\|$, described in Section 4.4.2. Specifically, $[\mathbf{h}_{j-1}]_j = 0$ results in $s_j = 0$ and $c_j = 1$ in Equations (4.37) and (4.39), which means that the rotation matrices Ω_j, as defined in Equation (4.33) is the identity matrix. This results in the residual being 0, which means that the then found set of vectors $\mathbf{h}_j$ leads to an exact solution x_m.

A disadvantage of a set amount of iteration is that it could be difficult to find a value of m that ensures a solution is found that satisfies the required accuracy, but does not require to much computation and memory storage. Since not much is known on what a good choice for this is, it is desired to use another condition for the convergence. The simplest way to add a convergence condition would be to compute the approximate solution $\mathbf{x}$ at certain intervals within the iteration and check for convergence. There however are other ways to check the convergence of the algorithm, which can be beneficial either in computational cost or accuracy.

A simple method for this is to make use of restarting. This is done by making sure the chosen value of m is small and repeating the algorithm multiple times. So instead of computing the approximate solution at intervals within the iteration, the iteration is stopped when $\mathbf{x}$ is computed, which is compared to a convergence condition. If this convergence condition is satisfied, the algorithm is stopped and the solution is returned. If this condition is not satisfied, a new iteration of the algorithm is started, where the initial guess of the new iteration is the found approximate solution of the previous iteration. This results in advantages both in storage and computation. Because at each iteration of the algorithm less vectors need to be stored, less storage is required than when using a larger value of m and the approximate solution is computed at certain intervals. Furthermore, the computation of $\mathbf{v}$ and $\mathbf{z}$ become more costly as j increases, which means that using a lower upper bound of j is advantageous.

4.4.4 Convergence of GMRES

Like with the methods observed earlier, it is desired to know if the GMRES method converges and how fast this convergence happens. The first important observation regarding the convergence of the method is that at each step the residual norm is minimized in a subspace $\mathcal{K}_m$. For an iteration that minimizes the residual, it is known that, if matrix A is positive definite,

$$\|\mathbf{r}_{k+1}\|_2 \leq \sqrt{1 - \frac{\mu^2}{\sigma^2}}\|\mathbf{r}_k\|_2,$$

where $\mu = \frac{1}{2}\lambda_{min}(A + A^T)$ and $\sigma = \|A\|_2$. This means that if A is a positive definite matrix, then the GMRES method converges for any m.

Next, it is also possible to establish a relation similar to the relation in Equation (4.26), which tells that a solution obtained from m steps of the Conjugate Gradient method minimizes the error for a polynomial of degree m. For the GMRES method, the solution obtained after m steps of the method minimises the 2-norm of the residual in $\mathbf{x}_0 + \mathcal{K}_m$. Since $\mathcal{K}_m$ contains all vectors of the form $\mathbf{x}_0 + q(A)\mathbf{r}_0$, where q is a polynomial with maximum degree $m-1$, the solution is of the form $\mathbf{x}_m = \mathbf{x}_0 + q_m(A)\mathbf{r}_0$, where $q_m(A)$ is the polynomial that minimizes the residual. Then it is observed that for the residual at step m,

$$\begin{aligned}
\|\mathbf{r}_m\|_2 &= \|\mathbf{b} - A\mathbf{x}_m\|_2 \\
&= \|\mathbf{b} - A\mathbf{x}_0 - Aq_m(A)\mathbf{r}_0\|_2 \\
&= \|\mathbf{r}_0 - Aq_m(A)\mathbf{r}_0\|_2 \\
&= \min_{q \in \mathbb{P}_{m-1}} \|\mathbf{r}_0 - Aq(A)\mathbf{r}_0\|_2.
\end{aligned}$$

This immediately gives an important difference in the solution obtained from the CG method and the solution obtained from the GMRES method. The CG method minimises the error, the difference between the obtained solution and the exact solution, whereas the GMRES method minimizes the residual.

For the GMRES method it is more complicated to obtain a speed of convergence. A useful result can only be obtained if A is diagonalizable as $A = X\Lambda X^{-1}$, where X contains the eigenvectors of A and Λ contains the eigenvalues of A. For the convergence, Saad **(2003 [8, p. 217])** states in Corollary 6.33 that for a diagonalizable matrix $A = X\Lambda X^{-1}$, with eigenvalues located in ellipse $E(c, d, a)$, the residual norm obtained in m-step of GMRES satisfies

$$\|\mathbf{r}_m\|_2 \leq \kappa_2(X)\frac{C_m\left(\frac{a}{d}\right)}{|C_m\left(\frac{c}{d}\right)|}\|\mathbf{r}_0\|_2. \tag{4.44}$$

Here C_m are the Chebyshev polynomials, defined by

$$\begin{aligned}
C_0(x) &= 1 \\
C_1(x) &= x \\
C_{n+1} &= 2xT_n(x) - T_{n-1}(x).
\end{aligned}$$

The fraction $\frac{C_m\left(\frac{a}{d}\right)}{|C_m\left(\frac{c}{d}\right)|}$ can be expanded to and estimated as

$$\frac{C_m\left(\frac{a}{d}\right)}{|C_m\left(\frac{c}{d}\right)|} = \frac{\left(\frac{a}{d}+\sqrt{\left(\frac{a}{d}\right)^2-1}\right)^m + \left(\frac{a}{d}+\sqrt{\left(\frac{a}{d}\right)^2-1}\right)^{-m}}{\left(\frac{c}{d}+\sqrt{\left(\frac{c}{d}\right)^2-1}\right)^m + \left(\frac{c}{d}+\sqrt{\left(\frac{c}{d}\right)^2-1}\right)^{-m}} \tag{4.45}$$

$$\approx \left(\frac{a+\sqrt{a^2-d^2}}{c+\sqrt{c^2-d^2}}\right)^m. \tag{4.46}$$

The convergence rate obtained in Equation (4.44) can be simplified significantly if A is an SPD matrix. For SPD matrices there always exists a diagonalisation of A, $A = X\Lambda X^{-1}$ such that X is an orthogonal matrix, which means that $\kappa_2(X) = 1$. Furthermore, it is known that the eigenvalues of an SPD matrices are real, which means that the ellipse in which the eigenvalues are enclosed can be given by $E(\frac{\lambda_{\max}+\lambda_{\min}}{2}, \frac{\lambda_{\max}-\lambda_{\min}}{2}, \frac{\lambda_{\max}-\lambda_{\min}}{2})$. From this the estimate in Equation (4.46) becomes

$$\frac{C_m\left(\frac{a}{d}\right)}{C_m\left(\frac{c}{d}\right)} \approx \frac{\lambda_{\max}-\lambda_{\min}}{\lambda_{\max}+\lambda_{\min}+4\sqrt{\lambda_{\max}\lambda_{\min}}}. \tag{4.47}$$

So then an approximate speed of convergence of $\frac{\lambda_{\max}-\lambda_{\min}}{\lambda_{\max}+\lambda_{\min}+4\sqrt{\lambda_{\max}\lambda_{\min}}}$ is obtained. Unfortunately, the GMRES method does not provide a way to approximate the eigenvalues directly in a way in which the CG method does. This means that an estimate for the eigenvalues must be obtained in a different way.

4.5 Multigrid Methods

The methods discussed in the previous sections, which are based on finding solutions in a certain Krylov subspace, work well for smaller problems. The convergence in these methods, however, can become much slower if problems become larger. The combination of slower convergence and increased operations at each step may make the Krylov subspace methods less suitable for large systems. For the matrices derived from discretized PDE methods this means that size to which the mesh can be reduced is limited, as reducing the mesh size increases the size of the discretization matrices.

If it is desired to reduce the mesh size in a PDE problem, Multigrid methods can be very useful. These methods are specifically designed to solve discretized elliptic PDE problems and the rate of convergence does not depend on the mesh size desired to be used. Multigrid methods make use of discretizations of different sizes and combine this with iterations similar to the iterations described in Section 4.1. The basic iterations converge slowly in general, but certain parts of the iteration can converge very quickly. The eigenvectors of the iteration matrices can be divided into high- and low- frequency modes, depending on how quick the error is damped in the direction of an eigenvector. It is then observed that the high-frequency modes are the eigenvectors corresponding to the larger eigenvalues of the iteration matrix. Multigrid methods try to map the low-frequency modes to high-frequency modes by remapping the problem between coarser and finer meshes.

This section discusses how problems can be remapped between coarse and fine meshes and then uses this to establish two types of multigrid methods.

4.5.1 Prolongation and Restriction

To make use of methods that are applied to different levels of grid sizes, it is required to be able to transform a problem to another grid size. For this only the transfer operation between two mesh sizes, a fine mesh Ω_h and a coarse one Ω_H, with $H = 2h$, for a one-dimensional problem is observed. So if Ω_h has n grid points, then Ω_H has $\frac{1}{2}n$ grid points. These transfer operations can be applied repeatedly to obtain transfers between finer or coarser meshes. Transfer operations for higher dimensional problems are obtained directly from the transfer operation of the one-dimensional problem, where in general for dimension d, Ω_H has $\left(\frac{1}{2}\right)^d n$ grid points. The transfer operations between mesh sizes are called prolongation and restriction, with prolongation taking a system from the coarse grid to the fine grid and restriction doing the reverse.

The prolongation operator maps vectors from the coarse grid to the fine grid,

$$I_H^h : \Omega_H \to \Omega_h.$$

So here a vector $\mathbf{x}_H \in \mathbb{R}^n$ is mapped to $\mathbf{x}_h \in \mathbb{R}^{2n}$, which is most commonly done through linear interpolation. This results in I_H^h being a $n \times 2n$ matrix, given by

$$I_H^h = \frac{1}{2}\begin{bmatrix} 1 & & & \\ 2 & & & \\ 1 & & & \\ & 1 & & \\ & 2 & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \\ & & & 2 \\ & & & 1. \end{bmatrix} \tag{4.48}$$

This makes sure that the elements in the coarse grid are mapped, $\mathbf{x}_H^i$, to the elements in the fine grid that correspond to the same grid point, $\mathbf{x}_H^{2i}$, and that the elements in the fine grid which were not present in the coarse grid, of the form $\mathbf{x}_H^{2i+1}$, are the average of the two neighbouring grid points. With linear interpolation, a higher dimensional prolongation operator can be obtained directly from the one-dimensional operator using tensor product. The two-dimensional interpolation prolongation matrix is for example obtained as

$$I_{H_{(2D)}}^h = I_H^h \otimes I_H^h \tag{4.49}$$

The restriction operator is the reverse of the prolongation operator, so it maps from the fine grid to the coarse grid,

$$I_h^H : \Omega_h \to \Omega_H.$$

A very simple way of doing this is by mapping the grid points in the fine grid directly to the corresponding grid points in the coarse grid. This gives

$$\mathbf{x}_i^H = \mathbf{x}_{2i}^h \tag{4.50}$$

and is called the injection operator. With the injection operator the information obtained in grid points of the fine grid, that do not directly correspond to a grid point of the coarse grid, is not used. To make sure this information is not discarded, the more commonly used full

weighting operator is defined. This operator is given by

$$I_H^h = \frac{1}{4}\begin{bmatrix} 1 & 2 & 1 & & & & \\ & & 1 & 2 & 1 & & \\ & & & & \ddots & & \\ & & & & 1 & 2 & 1 \end{bmatrix} \tag{4.51}$$

and makes sure every element in the coarse grid is obtained from a weighted average of the corresponding element in the fine grid and its neighbours. It is observed that this restriction operator is a scaled transpose of the interpolation prolongation operator, $I_H^h = \frac{1}{2}\left(I_h^H\right)^T$. Applying the restriction and prolongation operator together to a vector does not result in the original vector as can be seen from

$$\begin{aligned}\left[I_h^H I_H^h \mathbf{x}^H\right]_i &= \frac{1}{4}\left(\left[I_h^H \mathbf{x}^h\right]_{2i-1} + 2\left[I_h^H \mathbf{x}^h\right]_{2i} + \left[I_h^H \mathbf{x}^h\right]_{2i+1}\right) \\ &= \frac{1}{4}\left(\frac{1}{2}\left[\mathbf{x}^H\right]_{i-1} + \frac{1}{2}\left[\mathbf{x}^H\right]_i + 2\left[\mathbf{x}^H\right]_{2i} + \frac{1}{2}\left[\mathbf{x}^H\right]_i + \frac{1}{2}\left[\mathbf{x}^H\right]_{i+1}\right) \\ &= \left(\frac{1}{8}[\mathbf{x}^H]_{i-1} + \frac{3}{4}[\mathbf{x}^H]_i + \frac{1}{8}[\mathbf{x}^H]_{i+1}\right).\end{aligned}$$

Like the interpolation prolongation operator, the weighted restriction operator can be obtained for higher dimensions from the one-dimensional operator using tensor products. The property of the weighted restriction operator being a scaled transpose of the interpolation prolongation operator then remains, with

$$I_H^h = \left(\frac{1}{2}\right)^d \left(I_h^H\right)^T \tag{4.52}$$

4.5.2 Geometric Multigrid

Geometric Multigrid (SMG) techniques involve solving a problem at different levels of grid size. Typically this is done by obtaining a solution on a coarse grid first, which captures the low-frequency components of the error well. This solution is then interpolated and possibly smoothed to obtain an initial guess for a finer grid, from which a solution to the same problem on a finer grid is obtained. The solution on the finer grid captures the high frequency components of the error. The process can be applied multiple times recursively to go back and forth between different levels of grid size.

The goal of SMG methods is to solve a problem on a fine grid with mesh size h, of the form

$$A_h \mathbf{x}^h = \mathbf{f}^h. \tag{4.53}$$

To apply SMG methods it is required to have a similar problem on a coarse grid with mesh size H,

$$A_H \mathbf{x}^H = \mathbf{f}^H. \tag{4.54}$$

A way to obtain the system on the coarse grid is by discretizing the problem used to derive the system on the fine grid on the coarse grid in the same way. Alternatively this can be done using Galerkin projection, where A_H and $\mathbf{f}^H$ are obtained using the prolongation and restriction operators described in Section 4.5.1 as

$$A_H = I_h^H A_h I_H^h, \qquad f^H = I_h^H f^h \tag{4.55}$$

After solving the problem with a certain grid size, the problem at a new grid size is constructed and the solution found on the previous grid size is projected to the new grid size. This projected solution is then used to obtain a new initial guess for the next iteration by performing smoothing steps on the projected solution. A chosen amount of smoothing steps ν is done to obtain the smoothed solution $\mathbf{x}_\nu^h$. This smoothing step is of the form

$$\mathbf{x}_{j+1}^h = \mathbf{x}_j^h + G_h(\mathbf{f} - A_h\mathbf{x}_j^h). \tag{4.56}$$

The smoothing matrix G_h depends on the type of iteration that is used, as it is the iteration matrix as given in Equation (4.3), but require some type of damping. For example the Richardson iteration with $G = \omega I$, weighted Jacobi iteration with $G = \omega D$ and SOR iteration with G from Equation (4.18) are commonly used. Like how a smoothed solution $\mathbf{x}_\nu^h$ is obtained, a smoothed residual and error can be obtained as

$$\mathbf{r}_\nu^h = (I - A_hG_h)^\nu \mathbf{r}_0^h$$
$$\mathbf{d}_\nu^h = (I - G_hA_h)^\nu \mathbf{d}_0^h$$

4.5.2.1 Two-grid cycle

A theoretical simple but in practice not very useful method of applying geometric multigrid is the two-grid cycle method. This makes use of two-grid sizes, h and H, and solves on the finer grid size by correcting and smoothing a solution obtained on the coarse grid. Starting with an initial guess $\mathbf{x}_0^h$ on the fine grid, this is done by smoothing the initial guess to $\mathbf{x}_{\nu_1}^h$ using the smoothing operation given in Equation (4.56) and obtaining the residual $\mathbf{r}^h = \mathbf{f}^h - A_h\mathbf{x}^h$ on the fine grid. This residual is transformed to the coarse grid and a correction factor $\boldsymbol{\delta}^H$ is obtained by solving

$$A_H\boldsymbol{\delta}^H = \mathbf{r}^H.$$

This is used to obtain a correct the previously used solution $\mathbf{x}_{\nu_1}^h$, by

$$\mathbf{x}^h = \mathbf{x}_{\nu_1}^h + I_H^h\boldsymbol{\delta}^H.$$

The obtained solution is then post-smoothed, by applying ν_2 smoothing steps, to obtain the result of one iteration of the two-grid cycle $\mathbf{x}_{\text{new}}^h$. What is important in this method is that it is never required to solve a system on the fine grid as the only time a linear system is solved is on the coarse grid.

If the system that is desired to be solved has $\mathbf{f} = 0$, the smoothing step described in Equation (4.56) can be written as $\mathbf{x}_{j+1}^h = S_h\mathbf{x}_j^h$, with $S_h = I - B_hA_h$. This means that applying the smoothing step ν times to $\mathbf{x}_0^h$ is the same as multiplying $\mathbf{x}_0^h$ by $(S_h)^\nu$. Using this the two-grid cycle can be written as one iteration if $\mathbf{f}_h = 0$, with

$$\mathbf{x}_{\text{new}}^h = (S_h)^{\nu_2}\left[I_h - I_H^hA_H^{-1}I_h^HA_h\right](S_h)^{\nu_1}\mathbf{x}_0^h. \tag{4.57}$$

4.5.2.2 V-and W-cycle

Although the two-grid cycle itself is not very useful in practice, it can be used to derive two methods that are more useful. These are called the V-cycle and W-cycle and make use of the process described for the two-grid cycle. In both methods the starting grid with mesh size h_n is too fine to obtain a solution, but there is a coarsest grid with mesh size h_0, where it is possible to solve a transformed problem. To get to the coarse problem, the problem is repetitively coarsened to new grid sizes h_i, with $h_i = 2h_{i+1}$. The V-cycle uses a process that consists of

recursively applied two-grid cycles. The residual on a fine mesh size h_i is calculated in the same way and this residual is coarsened to a coarser grid to obtain $\mathbf{r}^{h_{i-1}}$. If the newly obtained grid size h_{i-1} is still too fine to solve the system, another cycle is started, with matrix $A_{h_{i-1}}$, initial guess $\mathbf{x}_0^{h_{i-1}} = \mathbf{0}^{h_{i-1}}$ and right hand side $\mathbf{f}^{h_{i-1}} = \mathbf{r}^{h_{i-1}}$. This is repeated until the coarse enough grid size h_0 is reached. At grid size h_0 the correction term $\boldsymbol{\delta}^{h_0}$ is calculated and $\mathbf{x}_{\text{new}}^h$ is obtained in the same way as is done in the two-grid cycle. The W-cycle uses a similar process, but does not obtain $\mathbf{x}_{\text{new}}^h$ directly when $\boldsymbol{\delta}^{h_0}$ is obtained if a coarse enough grid is reached. Instead it uses it to first go back one grid level, to a grid with mesh size h_1, and applies the correction term to the solution at that grid level. The new solution is then used as an initial guess to perform another two-grid cycle between h_1 and h_0, before going up one more level. This is repeated at each level of grid size, so before advancing to a higher level, the cycle first goes back to the lowest level. This process of going back and forth between layers can be repeated, depending on a parameter γ.

In general the multigrid method of the form of a V- or W- cycle depends on four parameters. Two of these, ν_1 and ν_2, determine the amount of smoothing steps, with ν_1 influencing the amount of pre-smoothing and ν_2 influencing the amount of post-smoothing. The parameter n depends on the total number of levels of grid size that is used, with the finest grid size being given by h_n. The last parameter γ influences the amount of two-grid cycles applied before advancing to higher layer. $\gamma = 1$ corresponds to the V-cycle, larger values of γ correspond to W-cycles.

4.5.3 Algebraic Multigrid

Geometric multigrid methods requires knowledge about the underlying mesh, which commonly is not available. It also is challenging to apply geometric multigrid methods to more complicated meshes, such as higher dimensional or non-rectangular meshes, which are commonly used in FEM based methods. Algebraic multigrid (AMG) uses the same Galerkin approach of interpolation and prolongation as geometric multigrid methods, but defines these operators only using coefficient matrix A. It thus does not require knowledge on the grid to which it is applied. To apply algebraic multigrid methods, it is required to have a matrix A that is positive definite.

To apply AMG to a fine problem

$$A_h \mathbf{x}^h = \mathbf{f}^h, \tag{4.58}$$

where h denotes the fine mesh, a way to transform the fine problem to a coarse problem is required. The fine problem finds a solution $\mathbf{x}^h \in X_h \subseteq \mathbb{R}^n$, the coarse problem

$$A_H \mathbf{x}^H = \mathbf{f}^H, \tag{4.59}$$

finds a solution $\mathbf{x}^H \in X_H \subseteq \mathbb{R}^m$. Here the dimension of the coarse problem is much lower than the dimension of the fine problem, with the dimension of the fine problem usually being around 2^d, with d the dimension of the physical problem, times as large as the dimension of the coarse problem. To transform this problem, it is required to find a way to coarsen the grid from X_h to X_H and define a restriction operator I_H^h.

Coarsening the grid is typically done by taking a subset of the nodes of the fine grid. Letting $F = (V_F, E_F)$ denote the graph of nodes on the fine grid and $C = (V_C, E_C)$ the graph of nodes of the coarse grid we have $V_C \subset V_F$. The way in which the coarse grid is obtained can be done in several ways, but must satisfy the requirements of having a good representation of smooth functions, the coarse problem being much smaller than the fine problem and strong couplings not being lost. A strong coupling is a couple of nodes i and j in the fine problem that have $\frac{[A_h]_{ij}}{[A_h]_{ii}} > \sigma$ for a predetermined value of σ. This coupling is maintained by ensuring that one

of the nodes is in C or one of the nodes is strongly coupled to a node in C. A simple and commonly used way of coarsening the grid is by letting V_C be an independent set of F.

The restriction operator is used to transform A_h and f_h to A_H and f_H, in the same way as it was done for other multigrid methods, by Equation (4.55). Thus it is given that $I_H^h \in \mathbb{R}^{n\times m}$. I_H^h must however now be defined algebraically from A_h, as it is not possible to use knowledge of the grid as it was done in Section 4.5.1. The prolongation operator is obtained directly from the transform of the restriction operator,

$$I_H^h = \left(I_h^H\right)^T. \tag{4.60}$$

It is desired to define the restriction operator in such a way that smoothness is maintained. To do this, for a coarse node $i \in V_C$, the nodes connected to i, nodes such that $[A]_{ij} \neq 0$, are divided in three groups. The first group, denoted by C_i is the set of nodes connected to i that are coarse nodes as well. The second group, denoted by F_i^s, is strongly connected to i, so for $j \in F_i^s$, $\frac{[A_h]_{ij}}{[A_h]_{ii}} > \sigma$. The last group, denoted by F_i^w, is weakly connected to i, so for $j \in F_i^s$, $\frac{[A_h]_{ij}}{[A_h]_{ii}} \leq \sigma$. With this a smooth error is defined as a $\mathbf{s} \in \mathbb{R}^n$, such that for smoothing matrix S_h,

$$\|S_h\mathbf{s}\|_A \approx \|\mathbf{s}\|_A.$$

From this it is obtained that $(A\mathbf{s}, \mathbf{s}) \approx 0$, so $A\mathbf{s} \approx 0$. For row i of A, this means

$$[A]_{ii}\mathbf{s}_i \approx -\sum_{j\neq i}[A]_{ij}\mathbf{s}_j = -\sum_{j\in C_i}[A]_{ij}\mathbf{s}_j - \sum_{j\in F_i^s}[A]_{ij}\mathbf{s}_j - \sum_{j\in F_i^w}[A]_{ij}\mathbf{s}_j. \tag{4.61}$$

This can be worked out to the relation $\mathbf{s} = \sum_{j\in C_i} w_{ij}\mathbf{s}_j$ **(Saad, 2003 [8, p. 459])**, with

$$w_{ij} = -\frac{[A]_{ij} + \sum_{k\in F_i^s}\frac{[A]_{ik}[A]_{kj}}{\delta_k}}{[A]_{ii} + \sum_{k\in F_i^w}[A]_{ik}}$$
$$\delta_k = \sum_{l\in C_i}[A]_{jl}.$$

This means that elements of the restriction operator are obtained as

$$\left[I_H^h\right]_{ij} = \begin{cases} 1 & i \in V_C, i = j \\ 0 & i \in V_C, i \neq j \\ w_{ij} & i \notin V_C, j \in V_C \\ 0 & i \notin V_C, j \notin V_C \end{cases} \tag{4.62}$$

Using this restriction operator and the corresponding prolongation operator, the methods shown for SMG, such as the two-grid cycle and the V-and W-cycle can be applied with AMG as well.

4.5.4 Convergence

To observe the convergence of a multigrid method, the results described by Saad **(2003 [8, p. 453])** are used, where a simple two-grid cycle is observed. The convergence properties of more complicated grid cycles can be obtained from expanding the results found here. Let S_h describe the smoothing process used and let T_h^H be the coarse grid correction operator,

$$T_h^H = I_h - I_H^h A_H^{-1} I_h^H A_h.$$

Both operators were used in a similar way in Equation (4.57). The smoothing operator can be assumed to satisfy the so-called smoothing property.

$$\left\|S_h e^h\right\|_{A_h}^2 \leq \left\|e^h\right\|_{A_h}^2 - \alpha \left\|Ae^h\right\|_{D^{-1}}^2 \quad \forall \quad e^h \in \Omega_h.$$

The coarse grid correction operator satisfies the approximation property, which states

$$\min_{u_H \in \Omega_H} \left\|e^h - I_H^h e^H\right\|_D^2 \leq \beta \left\|e^h\right\|_{A_h}^2 .$$

For a SPD original matrix A_h and prolongation and restriction operators satisfying a relation of the form of Equation (4.52), these two properties can be used to ensure convergence of the two-grid cycle. If the operators are such that they have both the smoothing property and the approximation property, then the two-grid cycle converges if $\alpha \leq \beta$. Furthermore, the operation of applying both the smoothing operator and coarse grid correction operator can be bounded in norm

$$\|S_h T_h^H\|_{A_h} \leq \sqrt{1 - \frac{\alpha}{\beta}}.$$

4.6 Block Methods and Domain Decomposition

In problems arising from discretized partial differential equations, it is often possible to divide the problem into multiple subproblems. This can be most advantageous if the domain is of a complicated shape, but can be divided into less complicated subproblems. Dividing the problem into p smaller problems is done by dividing the domain into p covering subsets. This results in the coefficient matrix A being divided into p^2 smaller matrices, each corresponding to the interactions within a subset of the domain or between two different subsets of the domain. This gives

$$A = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1p} \\ A_{21} & A_{22} & \cdots & A_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ A_{p1} & A_{p2} & \cdots & A_{pp} \end{bmatrix}. \tag{4.63}$$

Here the blocks on the diagonal, A_{ii}, correspond to interactions within domain i, other blocks, A_{ij} correspond to interactions between domain i and j. Figure 4.1a provides an illustration of an example of a cover that partitions a square domain, on which a finite difference approximation is applied, that is decomposed into three smaller domains. Figure 4.1b gives an illustration of how the corresponding matrix is divided into blocks.

Like how the coefficient matrix is divided into blocks, the right-hand side vector $\mathbf{b}$ and solution $\mathbf{x}$ are divided into smaller vectors, each corresponding to one of the subdomains of the division:

$$\mathbf{x} = \begin{pmatrix} \boldsymbol{\xi}_1 \\ \boldsymbol{\xi}_2 \\ \vdots \\ \boldsymbol{\xi}_p \end{pmatrix}, \qquad \mathbf{b} = \begin{pmatrix} \boldsymbol{\beta}_1 \\ \boldsymbol{\beta}_2 \\ \vdots \\ \boldsymbol{\beta}_p \end{pmatrix}. \tag{4.64}$$

The way of dividing $\mathbf{x}$ and $\mathbf{b}$ directly corresponds to the division of A, in a way such that

$$\boldsymbol{\beta}_i = \sum_{j=1}^{p} A_{ij} \boldsymbol{\xi}_j. \tag{4.65}$$

For the simple example in Figure 4.1 $\mathbf{x}$ and $\mathbf{b}$ are divided into three smaller vectors, each with $\boldsymbol{\xi}_i, \boldsymbol{\beta}_i \in \mathbb{R}^6$.

The covering partition shown here is done in a very simple way, by cutting certain edges in the graph that describes the domain. More generally, such a cover can be described by either a set-partition or a set-decomposition. A set-partition decomposes the variable set $S = \{1, 2, \dots n\}$ into disjoint subsets, $S_1, S_2 \dots S_p$. With these disjoint subsets, submatrices and subvectors of A, $\mathbf{x}$ and $\mathbf{b}$ are defined through these same partitions. This is done by defining the submatrices of A and subvectors of $\mathbf{x}$ and $\mathbf{b}$ as

$$\begin{aligned} A_{ij} &= [A]_{S_i,S_j} \\ \boldsymbol{\xi}_i &= [\mathbf{x}]_{S_i} \\ \boldsymbol{\beta}_i &= [\mathbf{b}]_{S_i} \end{aligned} \tag{4.66}$$

A set-decomposition allows for overlap and thus only requires the union of the subsets to be a cover of the original full set, so it defines subsets $S_1, S_2 \dots S_p$ such that

$$\begin{aligned} S_i &\subseteq S \\ \bigcup_{i=1,\cdots p} S_i &= S. \end{aligned}$$

For covers that allow overlap, it is required to define the submatrices A_{ij} and subvectors $\boldsymbol{\xi}_i$ and $\boldsymbol{\beta}_i$ in a way that is able to deal with the overlapping elements.

For a particular subset S_i, of size n_i, given by

$$S_i = \left\{ m_i^1, m_i^2, m_i^3, \cdots m_i^{n_i} \right\},$$

two matrices can be defined,

$$\begin{aligned} V_i &= \begin{bmatrix} \mathbf{e}_{m_i^1} & \mathbf{e}_{m_i^2} & \cdots & \mathbf{e}_{m_i^{n_1}} \end{bmatrix} \\ W_i &= \begin{bmatrix} \eta_{m_i^1}\mathbf{e}_{m_i^1} & \eta_{m_i^2}\mathbf{e}_{m_i^2} & \cdots & \eta_{m_i^{n_1}}\mathbf{e}_{m_i^{n_1}} \end{bmatrix}, \end{aligned} \tag{4.67}$$

which are used to define the submatrices A_{ij}. Here $\eta_{m_i^2}$ are weight factors, such that

$$W_i^T V_i = I.$$

Using the matrices W_i and V_j, obtained using the indices in subsets S_i and S_j, the submatrices $A_{ij} \in \mathbb{R}^{n_i \times n_j}$ are defined as

$$A_{ij} = W_i^T A V_j. \tag{4.68}$$

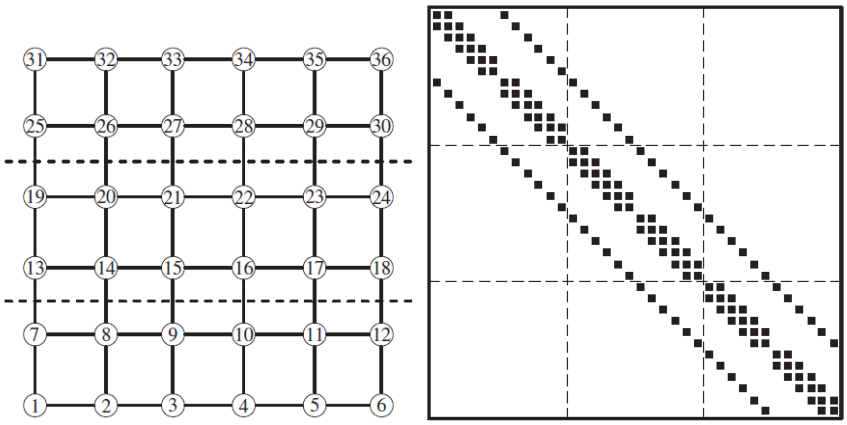
The subvectors corresponding to indices in a subset S_i are obtained as

$$\begin{aligned} \boldsymbol{\xi}_i &= W_i^T \mathbf{x} \\ \boldsymbol{\beta}_i &= W_i^T \mathbf{b}, \end{aligned}$$

from which $\mathbf{x}$ can be obtained as

$$\mathbf{x} = \sum_{i=1}^{p} V_i \boldsymbol{\xi}_i.$$

It is observed that defining the submatrices and subvectors using matrices V_i and W_i for disjoint subsets S_i leads to the same submatrices and subvectors obtained from directly taking the elements with corresponding indices, as was done in Equation (4.66).



(a) (b)

Figure 4.1: (a) 6x6 Domain of finite difference approximation decomposed into 3 subdomains (b) with corresponding matrix divided into 9 blocks **(Saad, 2003 [8, p. 110])**

4.6.1 Block relaxations for Basic Iterative Methods

For the Basic Iterative Methods discussed in Section 4.1 a variant can be described that makes use of blocks as described by Equation (4.66). Here a splitting of A of the form $A = D - E - F$ is used, similar to the one described in Equation (4.4). However, the matrices are now described using the blocks obtained from A by Equation (4.66) instead of diagonal, lower and upper elements. The matrices D, E and F are described by

$$D = \begin{bmatrix} A_{11} & O & \cdots & O \\ O & A_{22} & \cdots & O \\ \vdots & \vdots & \ddots & \vdots \\ O & O & \cdots & A_{pp} \end{bmatrix}, \quad E = -\begin{bmatrix} O & O & \cdots & O \\ A_{21} & O & \cdots & O \\ \vdots & \vdots & \ddots & \vdots \\ A_{p1} & A_{p2} & \cdots & O \end{bmatrix}, \quad F = -\begin{bmatrix} O & A_{12} & \cdots & A_{1p} \\ O & O & \cdots & A_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ O & O & \cdots & O \end{bmatrix}.$$

This splitting can be used to define the iterations of the basic iterative methods, in exactly the same way as the splitting described by Equation (4.4). This means that the Jacobi iteration is again described by Equation (4.14), Gauss-Seidel by Equation (4.16) and SOR by Equation (4.17), but with the newly defined matrices D, E and F.

Note that this can only be applied directly for covers without overlap. For covers that allow for overlap, matrices D, E and F would not be of the same size as A making it not possible to apply the direct iteration schemes. This means that instead an iteration is defined using the element-wise iterations for the basic iteration methods. For the Jacobi and Gauss-Seidel iteration these are given in Algorithm 4.4 and 4.5.

4.6.2 Domain Decomposition

Using the idea of dividing the problem into multiple subproblems, a more formal method can be described. This method is called Domain Decomposition and makes use of a decomposition between so called subdomain nodes, located in multiple subdomains, and interface nodes, that correspond to interaction between subdomains. This means that, when working with p subdomains, the set of indices corresponding to model nodes, $S = \{1, \ldots, n\}$, is divided into a cover of $p + 1$ subsets. The first p subsets, $S_1, \ldots, S_p$ each correspond to one subdomain. The last subset, S_C, corresponds contains the indices of the interface nodes.

Algorithm 4.4: Block Jacobi Iteration **(Saad, 2003 [8, p. 112])**

Data: Coefficient Matrix A, Right-hand side Vector $\mathbf{b}$, Initial Guess $\mathbf{x}_0$, Partition matrices V_i, W_i, A_{ii} (as in Equations (4.67) and (4.68))
Result: Solution $\mathbf{x}$, Residual $\mathbf{r}$
Initialize $k = 0$
while *convergence condition not satisfied* **do**
  Set $r_k = b - Ax_k$
  for $i = 1 \cdots p$ **do**
    Solve $A_{ii}\delta_i = W_i^T r_k$
    Set $x_{k+1} = x_k + V_i\delta_i$
  end
  $k = k + 1$
end

Algorithm 4.5: Block Gauss-Seidel Iteration **(Saad, 2003 [8, p. 112])**

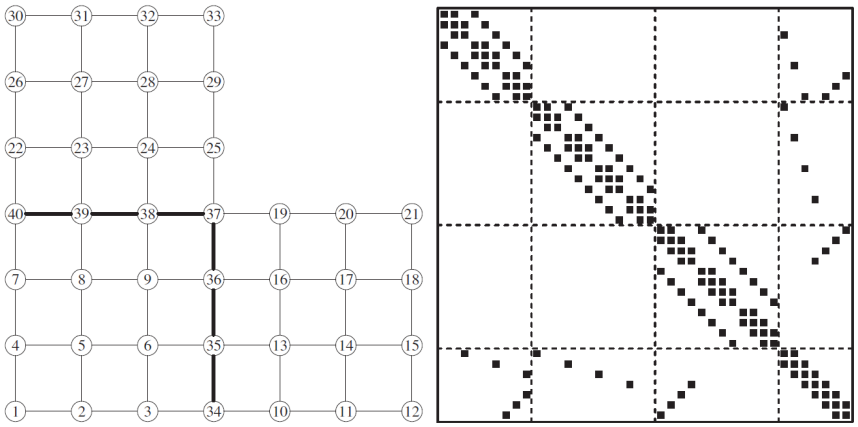
Data: Coefficient Matrix A, Right-hand side Vector $\mathbf{b}$, Initial Guess $\mathbf{x}_0$, Partition matrices V_i, W_i, A_{ii} (as in Equations (4.67) and (4.68))
Result: Solution $\mathbf{x}$, Residual $\mathbf{r}$
while *convergence condition not satisfied* **do**
  for $i = 1 \cdots p$ **do**
    Solve $A_{ii}\delta_i = W_i^T(b - Ax)$
    Set $x = x + V_i\delta_i$
  end
end

Again the matrix-vector system can be split according to these subsets, like how it was done in Equations (4.63) and (4.64). Since there is no interaction between different subdomains, only between the subdomains and the interface nodes, it is observed that any submatrix A_{ij} is the zero matrix, if $i \neq j$ and neither i or j is $p + 1$. This results in a system of the form

$$\begin{bmatrix} B & E \\ F & C \end{bmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} = \begin{pmatrix} \mathbf{f} \\ \mathbf{g} \end{pmatrix}, \tag{4.69}$$

where $\begin{bmatrix} B & E \\ F & C \end{bmatrix}$, $\begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix}$ and $\begin{pmatrix} \mathbf{f} \\ \mathbf{g} \end{pmatrix}$ each correspond to $A, \mathbf{x}$ and $\mathbf{b}$ in the original system, as described in Equation (4.1). The submatrices, $B \in \mathbb{R}^{(n-|S_C|)\times(n-|S_C|)}$, $C \in \mathbb{R}^{|S_C|\times|S_C|}$, $E \in \mathbb{R}^{(n-|S_C|)\times|S_C|}$ and $F \in \mathbb{R}^{|S_C|\times(n-|S_C|)}$, each describe a specific part of the interactions between nodes. B is a diagonal block matrix, with blocks B_i, each describing the interactions between nodes of indices in subset S_i. E and F both are made up of submatrices, E_i and F_i, which describe the interaction between subdomain i and the interface nodes. C describes the interaction between interface nodes. The solution and right-hand side vectors, $\begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix}$ and $\begin{pmatrix} \mathbf{f} \\ \mathbf{g} \end{pmatrix}$, both consist of a part corresponding to the subdomain nodes, $\mathbf{f}$, $\mathbf{x} \in \mathbb{R}^{n-|S_C|}$, and a part corresponding to the interface nodes, $\mathbf{g}$, $\mathbf{y} \in \mathbb{R}^{|S_C|}$.

Figure 4.2a, shows an example of a domain of a more complicated shape being split into 3 rectangular subdomains and a group of interface nodes. Here it is observed that the subdomain subsets are $S_1 = \{1, \ldots 9\}$, $S_2 = \{10, \ldots 21\}$ and $S_3 = \{22, \ldots 33\}$ and the interface subset is $S_C = \{34, \ldots 40\}$. Figure 4.2b shows an impression of the corresponding matrix, with the different blocks marked. As can be seen, there are no interactions between different subdomains.



(a) (b)

Figure 4.2: (a) L-shaped domain of finite difference approximation decomposed into 3 subdomains and interface nodes (b) with corresponding matrix divided into blocks corresponding to Equation (4.69), this matrix is referred to as an arrowhead matrix. **(Saad, 2003 [8, p. 473])**

The decomposition shown in Figure 4.2a is one way of decomposing the domain, but depending on certain choices, this decomposition can be done in several ways. The first thing that must be decided is the amount of subdomains and the size of these subdomains. The next choice that has to be made is whether overlap of subdomains is allowed. Allowing subdomains to overlap can require changes to the block matrix, to ensure there are no values in the parts that are expected to be zero. Alternatively, alterations can be made to the methods used, to make sure this is dealt with.

A last important choice is how the partitioning is done. For this three possibilities are commonly used, the first of which, element-base partitioning, was used in Figure 4.2a. This method of partitioning makes a split based on elements and does not allow an element to be split between two subdomains. The second method, edge-based partitioning, makes a split in a way in which no edges are split between two different subdomains. The last method, vertex-based partitioning, splits the original set of vertices into subsets of vertices, without restrictions on elements or edges.

Because of the format of the matrix, the most complicated part in solving the system in Equation (4.69) is finding the solution for the part that corresponds to the interface nodes, $\mathbf{y}$. For this solution, it is observed that

$$S\mathbf{y} = \mathbf{g} - FB^{-1}\mathbf{f}, \tag{4.70}$$

where S is the so called Schur complement matrix $S = C - FB^{-1}E$. This Schur complement matrix satisfies the properties of being non-singular if A is non-singular and of being SPD if A is SPD as well **(Saad, 2003 [8, p. 476])**. This computation can be complicated, as it requires B^{-1}, it however is less complicated than solving the original system.

If this $\mathbf{y}$ is obtained, the solutions in the subdomains, $\mathbf{x}_i$, can easily be obtained from

$$B_i\mathbf{x}_i = \mathbf{f}_i - E_i\mathbf{y}_i.$$

This can be solved using a direct or iterative solver, depending on the size and complexity of the resulting problem.

5 Preconditioning

Krylov subspace methods like the CG and the GMRES method are theoretically well-working methods, they however can lack in speed of convergence when applied to practical problems. Although these solvers will in theory always converge to a solution, this convergence can be very slow. To improve the efficiency of these iterative methods one can use preconditioning. Preconditioning a system is done by transforming the system into another system that has the same solution but on which the performance of an iterative solver is improved. The way to improve the performance is usually by making sure the new system is more well-conditioned than the original system, which improves the speed of convergence. The typical way to transform the system is by multiplying it by a preconditioning matrix M in one of three different ways. The first way to transform a system is by left multiplication of the system, leading to

$$M^{-1}A\mathbf{x} = M^{-1}\mathbf{b}. \tag{5.1}$$

The second is by applying M to the right, leading to

$$AM^{-1}\mathbf{u} = \mathbf{b}, \qquad \mathbf{x} = M^{-1}\mathbf{u}. \tag{5.2}$$

The last way to apply a preconditioning matrix is a combining the first two methods. If a preconditioning matrix can be split in two triangular matrices with $M = M_L M_R$, then the left and right method can be combined as

$$M_L^{-1}AM_R^{-1}\mathbf{u} = M_L^{-1}\mathbf{b}, \qquad \mathbf{x} = M_R^{-1}\mathbf{u}. \tag{5.3}$$

The preconditioning matrix M can be defined in several ways, but has a few minimal requirements it must satisfy. First, M obviously has to be non-singular, as it has to be possible to make use of the inverse of M. Secondly, as it is desired to speed up the convergence of the system, solving $M\mathbf{x} = \mathbf{b}$ has to be computationally inexpensive. Lastly, it is desired to make sure M is similar to the original matrix A, to make sure M^{-1} is an approximation of A^{-1}. This causes $M^{-1}A$ to be close to the identity matrix, which means that $\kappa_2(M^{-1}A) \approx 1$, as the condition number of the identity matrix is 1.

This chapter first discusses ways of applying preconditioning to the Conjugate Gradient method in Section 5.1 and the GMRES method in Section 5.2. After this ways to actually find preconditioner matrices are discussed in Sections 5.3, 5.4 and 5.5.

5.1 Preconditioned Conjugate Gradient

An important property of matrices to which CG usually is applied is the fact that they are required to be SPD. As a preconditioning matrix M is chosen in a way such that it is similar to A, which means that it is required for M to be SPD as well. With this M, preconditioning can be applied directly, using one of the methods described above. This however may cause problems in the application of CG to the obtained systems, as the obtained matrices $M^{-1}A$ in Equation 5.1 and AM^{-1} in Equation 5.2 are not necessarily SPD. This is only the case if A and M^{-1} commute, which they in general do not do, as this would mean that the left and right

preconditioned systems are the same. It is thus important to apply a preconditioning matrix in a way in which the obtained system is a SPD system, so to make sure CG can be applied.

A simple way to make sure a preconditioned system is preserved is using a split preconditioned system

$$L^{-1}AL^{-T}\mathbf{u} = L^{-1}\mathbf{b}, \qquad x = L^{-T}\mathbf{u}, \tag{5.4}$$

where L is obtained from an incomplete Cholevski factorisation of M, so $M = LL^T$. The matrix in this system, $L^{-1}AL^{-T}$, is SPD, as for any SPD matrix A and non-singular B, BAB^T is SPD. This means that it is possible to apply CG to this obtained system.

It can however be undesirable to compute the Cholevski factorisation of the preconditioning matrix. This means that it is desired to develop an alteration to the CG method that takes the preconditioner matrix into account. For this it is observed that $M^{-1}A$ is not necessarily self-adjoint with the Euclidian inner product, but it is self-adjoint with respect to the M- inner product, as

$$(M^{-1}A\mathbf{x}, \mathbf{y})_M = (A\mathbf{x}, \mathbf{y}) = (\mathbf{x}, A\mathbf{y}) = (\mathbf{x}, MM^{-1}Ay) = (\mathbf{x}, M^{-1}A\mathbf{y})_M.$$

From this an alternative CG method is obtained that replaces Euclidian inner product by M-inner products. This results in iterates that are obtained in a similar way as described in Section 4.2.1, with a new residual for the preconditioned system described by $\mathbf{z}_j = M^{-1}\mathbf{r}_j$, obtained from the residual. This residual is used to alter the scalars α_j from Equation 4.23 and β_j from Equation 4.25, along with the calculation of the vectors $\mathbf{p}_j$ that span the Lanczos space. The new way in which these are computed is by

$$\begin{aligned}
\alpha_j &= \frac{(\mathbf{z}_j, \mathbf{z}_j)_M}{(M^{-1}A\mathbf{p}_j, \mathbf{p}_j)_M} = \frac{(\mathbf{r}_j, \mathbf{z}_j)}{(A\mathbf{p}_j, \mathbf{p}_j)} \\
\beta_j &= \frac{(\mathbf{z}_{j+1}, \mathbf{z}_{j+1})_M}{(\mathbf{z}_j, \mathbf{z}_j))_M} = \frac{(\mathbf{r}_{j+1}, \mathbf{z}_{j+1})}{(\mathbf{r}_j, \mathbf{z}_j))} \\
\mathbf{p}_{j+1} &= \mathbf{z}_{j+1} + \beta_j \mathbf{p}_j.
\end{aligned}$$

The second variants to compute α_j and β_j are what would be used in practice, as these omit the computation of the M-inner products. Saad **(2003 [8, p. 277])** provides a full overview of the newly obtained algorithm for this Preconditioned CG method. In a similar way a preconditioned CG method for the right preconditioned system can be composed, using M^{-1} inner products. This however ends up giving the exact same iterates as the left preconditioned system. Furthermore, applying CG to the split preconditioned system using the incomplete Cholevski decomposition also results in the same iterates as the left preconditioned CG method.

Another way in which preconditioning on the CG method can be done is using Eisenstat's implementation. In this implementation matrix A is split, using its symmetric property, in a diagonal matrix D_0 and triangle matrices E and E^T, by

$$A = D_0 - E - E^T. \tag{5.5}$$

Using the matrices from the splitting, preconditioned CG is applied to

$$\hat{A}\mathbf{u} = (D - E)^{-1}\mathbf{b}$$

with

$$\hat{A} = (D - E)^{-1}A(D - E^T)^{-1}, x = (D - E^T)^{-1}\mathbf{u}$$

and $M^{-1} = D$ as a preconditioner matrix. Here D is a diagonal matrix that is not necessarily the same as D_0, but is often related to it. This method generally requires $3n + 2\text{nnz}(A)$ operations,

while the regular preconditioned CG method requires $4\text{nnz}(A) - n$ operations. This means that Eisenstat's method is computationally less costly than the regular method. However, a downside is that only a single special preconditioner can be used.

5.2 Preconditioned GMRES

GMRES has no requirements for the system which is desired to be solved, other than the coefficient matrix being non-singular. This means that applying preconditioning is relatively straight forward. The three variants of preconditioning as discussed in Equations 5.1-5.3 can all be applied directly to obtain three Preconditioned GMRES (P-GMRES) methods. For right preconditioning a special flexible variant exists, that makes use of a different preconditioning matrix at each iteration.

Left P-GMRES is simply done by applying the GMRES method as described in Section 4.4 to the system in Equation 5.1. This means that the Krylov subspace that is constructed by the GMRES method is of the form

$$\text{Span}\left\{\mathbf{z}_0, M^{-1}A\mathbf{z}_0, (M^{-1}A)^2\mathbf{z}_0, \dots (M^{-1}A)^{m-1}\mathbf{z}_0\right\}, \tag{5.6}$$

where $\mathbf{z}_0$ corresponds to the preconditioned initial residual $M^{-1}(\mathbf{b} - A\mathbf{x}_0)$. As this subspace is based on the residual of the preconditioned system, all residuals computed in the GMRES iteration correspond to preconditioned residuals $\mathbf{z}_m = M^{-1}(\mathbf{b} - A\mathbf{x}_m)$. This means that if it is desired to find the residual of the original system, without a preconditioner, it is required to multiply the obtained preconditioned residual $\mathbf{z}_m$ by M. This makes it more challenging to base a stopping condition on the actual residual and instead it is more convenient to use a stopping criterion based on the preconditioned residual. If A is close to an SPD matrix or is an SPD matrix itself, it is possible to use a preconditioning matrix M that is SPD. This results in an algorithm, which replaces the Euclidian inner products in the GMRES method by M-inner products in the same way as it was done for the preconditioned CG method. A problem that might arise when doing this is certain inner products becoming negative due to round-off errors.

In the same way right preconditioning can be used with the GMRES method. This results in applying the GMRES method to Equation 5.2, with a simple alteration to find the initial residual. Since $\mathbf{u}_0$ is commonly not available, $\mathbf{b} - AM^{-1}\mathbf{u}_0$ can not be computed. $\mathbf{x}_0$ however is available, so it is possible to compute the initial residual from $\mathbf{b} - A\mathbf{x}_0$. Applying GMRES to a right preconditioned system leads to solutions in the Krylov subspace given by

$$\text{Span}\left\{\mathbf{r}_0, AM^{-1}\mathbf{r}_0, (AM^{-1})^2\mathbf{r}_0, \dots (AM^{-1})^{m-1}\mathbf{r}_0\right\}. \tag{5.7}$$

Unlike left P-GMRES, right P-GMRES uses residuals of the original system, $\mathbf{r}_m = \mathbf{b} - A\mathbf{x}_m$. This can be more convenient when defining a stopping condition.

The third way to apply preconditioning to the GMRES method is to use split preconditioning. This is done by splitting the matrix M by LU-factorisation in matrices L and U such that $LU = M$. This results in a split preconditioned system

$$L^{-1}AU^{-1}\mathbf{u} = L^{-1}\mathbf{b}, \qquad \mathbf{x} = U^{-1}\mathbf{u}.$$

Applying GMRES results in residuals preconditioned by the left split, $\mathbf{r}_m = L^{-1}(\mathbf{b} - A\mathbf{x}_m)$.

The main difference between the three preconditioning methods is the way in which residuals are computed. This means that each method needs a stopping criterion that is slightly different. Determining a well-working stopping criterion based on the preconditioning matrix M can be

challenging if M is ill-conditioned. Between the left P-GMRES and right P-GMRES another connection can be made in the function they minimize. Left P-GMRES minimizes

$$\|M^{-1}\mathbf{b} - M^{-1}A\mathbf{x}\|_2,$$

whereas right P-GMRES minimizes

$$\|\mathbf{b} - AM^{-1}\mathbf{u}\|_2.$$

These two are only substantially different if M is ill-conditioned, which it is not expected to be. This means that it can be assumed that there is not massive difference in quality of performance between left and right P-GMRES.

5.3 BIM based Preconditioners

Now that ways in which preconditioners can be used to help improve iterative methods are established, it is necessary to find ways to define the preconditioning matrix M described above. The most simple way of obtaining a preconditioner matrix is from the basic iterative methods discussed in 4.1. These methods make use of an iteration of the form of Equation (5.8)

$$\mathbf{x}_{k+1} = G\mathbf{x}_k + \mathbf{f}, \tag{5.8}$$

where $G = I - M^{-1}A$, where M is obtained from some way of splitting $A = M - N$. This iteration solves $(I - G)\mathbf{x} = \mathbf{f}$, which is essentially the same as solving the left preconditioned system described in Equation 5.1. Thus, the matrix M obtained from splitting A can be used as a left preconditioning matrix. These exact same matrices where used to described the basic iterative methods in Section 4.1, which used an iteration of the form $I - M^{-1}A$, so they can be used as preconditioner matrices here as well.

For the Jacobi method the iteration matrix is given by $I - D^{-1}A$, where D consists of the elements on the diagonal of A. This means that the obtained preconditioning matrix is given by D. Preconditioning with $M = D$ is equivalent to scaling the matrix such that every diagonal element becomes one. In the same way the Gauss-Seidel iteration uses an iteration matrix of $I - (D - E)^{-1}A$, so then $M = D - E$ is used as a preconditioning matrix.

Preconditioning matrices obtained from damped methods such as the Damped Jacobi iteration and SSOR can be used as preconditioners as well. However, the choice of the damping parameter ω often is less important than it is in a fixed-point iteration. Generally, the most simple choice, $\omega = 1$, is used, which causes the damped Jacobi method to be exactly the same as the standard Jacobi method. For the SSOR iteration, which has preconditioning matrix

$$M = (D - \omega E)D^{-1}(D - \omega F),$$

this results in the Symmetric Gauss-Seidel preconditioning matrix,

$$M = (D - E)D^{-1}(D - F). \tag{5.9}$$

This is the same as using the LU-decomposition of A for preconditioning.**(Saad, 2003 [8, p. 299])**

5.4 ILU factorisation

Another way to define preconditioning matrices is using incomplete LU factorisation. This uses an approximation of the LU factorisation of A, with some residual matrix R, such that

$$A = LU + R. \tag{5.10}$$

Like the full LU factorisation of a matrix, methods for obtaining an ILU factorisation of a matrix are based on Gaussian elimination. The difference between the ILU factorisation and the full LU factorisation is that in the ILU factorisation certain positions are chosen in which the elements are dropped. For ILU factorisation a zero pattern set P, which is a subset of all off-diagonal elements, is defined, which is used to compute an L and U that satisfy this zero pattern. The zero pattern set can be defined in several ways, giving different ILU methods.

A general algorithm for the ILU method is provided in Algorithm 5.1. This algorithm is derived from the algorithm for computing the LU decomposition using Gaussian elimination, with the extra requirement of forcing certain elements to zero. Using the empty set as the zero pattern results in the full LU decomposition. The result of this algorithm is the upper matrix U, with the elements of L obtained during the intermediate steps.

Algorithm 5.1: ILU Factorisation **(Saad, 2003 [8, p. 303])**

Data: Coefficient Matrix A, Zero pattern $P \subset \{(i,j)|i \neq j; 1 \leq i,j \leq n\}$
Result: ILU decomposition L and U
Initialize $L = I_n$
for $(i,j) \in P$ **do**
  Set $[A]_{i,j} = 0$
end
for $k = 1, \ldots, n-1$ **do**
  for $i = k+1, \ldots, n$ **do**
    if $(i,k) \notin P$ **then**
      $[A]_{i,k} = \frac{[A]_{i,k}}{[A]_{k,k}}$
      for $j = k+1, \ldots, n$ **do**
        if $(i,j) \notin P$ **then**
          $[A]_{i,j} = [A]_{i,j} - [A]_{k,k} \cdot [A]_{k,j}$
        end
      end
    end
  end
end
Set L and U as the lower and upper triangles of A.

5.4.1 ILU(0)

A straight forward choice for the zero pattern P in the ILU factorisation is the zero pattern of the original matrix A. This provides us with the ILU(0) factorisation. This factorisation results in matrices L and U which have a zero pattern closely related to the zero pattern of A. In fact, the zero pattern of L is exactly the same as the zero pattern of A below the diagonal and the zero pattern of U is the zero pattern of A above the diagonal. For more efficient storage, it can be convenient to make use of the non-zero pattern of A instead of the zero pattern, denoted by

$NZ(A) = \{(i,j)|[A]_{i,j} \neq 0\}$, as A is expected to be a sparse matrix. Then, instead of checking whether a combination of indices (i,j) is not in P, it is checked whether $(i,j) \in NZ(A)$. With this non-zero pattern, it is observed that the elements of the residual $R = A - LU$ corresponding to indices in $NZ(A)$ are zero.

If A is such that the product of its strict-lower and strict-upper parts, E and F from Equation (4.4), only consists of diagonal elements and fill-in elements, the result of the ILU(0) factorisation is of the form

$$M = (D - E)D^{-1}(D - F). \tag{5.11}$$

The matrix D is a diagonal matrix, of which the elements are obtained from the ILU(0) factorisation. The obtained preconditioning matrix is similar to what is obtained from preconditioning with the Symmetric Gauss-Seidel method, as done in Equation (5.9). Note, however, here that the diagonal matrices are not the same. For the Symmetric Gauss-Seidel method, the diagonal matrix is the diagonal of matrix A, whereas the diagonal matrix used in ILU(0) is a diagonal matrix, that is such that the diagonal of the product M in Equation (5.11) is the same as the diagonal of A.

5.4.2 ILU(p)

Using the ILU(0) factorisation similar, more accurate factorisations can be described. To make the incomplete factorisation more accurate, it is desired to allow some fill-in, but only in those locations were it is expected to give a significant increase while limiting the required storage and amount of computations. A way to do this is to only allow fill-in up to some level p.

The level of fill-in is described by the non-zero patterns of recursively obtained incomplete LU-decompositions. Using the ILU(0) decomposition, matrices L_0 and U_0 are obtained, for which $NZ(L_0) = NZ(U_0) = NZ(A)$. Let this be denoted by $NZ_0(A)$. Now the product of L_0 and U_0 has a different non-zero pattern than A. Using L_0 and U_0, $NZ_1(A)$ is defined of the non-zero pattern of L_0U_0. With this newly obtained non-zero pattern, a new ILU decomposition, ILU(1) denoted by L_1 and U_1, is obtained by applying the ILU decomposition with $P = NZ_1(A)$. Using the obtained L_1 and U_1, a next non-zero pattern $NZ_2(A)$ can be defined, to which the same process is applied. Applying this process p times gives us a $NZ_p(A)$ and ILU decomposition matrices L_p and U_p. This decomposition is called the ILU(p) decomposition. For a specific element, the level of fill-in is the step at which the element is added to the non-zero set. So an element that is present in $NZ_p(A)$, but not in $NZ_{p-1}(A)$, has a fill-in level p.

Alternatively, the level of fill-in can be described using the Gaussian elimination process. For this, to each location (i,j) in matrix A a value for the level of fill-in, $lev_{i,j}$ is assigned. Initially, this is described by

$$lev_{i,j} = \begin{cases} 0 & \text{if} (i,j) \in NZ(A) \\ \infty & \text{if} (i,j) \notin NZ(A). \end{cases}$$

Then at each step of the Gaussian elimination process, $lev_{i,j}$ is updated if the element in location (i,j) is altered, so if the update $[A]_{i,j} = [A]_{i,j} - [A]_{i,k}[A]_{k,j}$ is applied. If this is done, the level of fill-in of (i,j) is updated by

$$lev_{i,j} = \min\{lev_{i,j}, lev_{i,k} + lev_{k,j} + 1\}.$$

This provides another way to determine the ILU(p) decomposition. By applying the complete Gaussian elimination process, but skipping any updates where $lev_{i,j}$ exceeds the maximum level of fill-in p, the same ILU(p) decomposition is obtained. The advantage of this is that only one

Gaussian elimination process is required, instead of the p required processes when using ILU decompositions of lower order to compute ILU(p).

In some special cases a clear relation between different levels of fill-in can be observed. For example, let A be a sparse matrix with non-zero values located in diagonals, so

$$NZ(A) = \{(i,j)|i = j + k, k \in K \subset \mathbb{N}\}.$$

Here K is a set of a limited size, so A has a few diagonals which are non-zero. The fill-in will be located around these diagonals. Specifically, fill-in of order p will be located in a diagonal that satisfies $i = j + k \pm p$, so p diagonals away from a diagonal that is include in $NZ(A)$. For general cases however, the amount of fill-in is unpredictable.

Most often in ILU(p) the dropped elements are simply discarded and not used for anything anymore. However, it can be advantageous to compensate for these dropped elements. The way to compensate for dropped elements is by subtracting all dropped elements in one row from the diagonal element in that same row. In Algorithm 5.1 this introduces an extra operation, applied after computing the L and U matrices, in which the sum of each row of obtained remainder matrix is subtracted from the diagonal element corresponding to that row,

$$[A]_{i,i} = [A]_{i,i} - \sum_{j=1} n[R]_{i,j}.$$

This ensures that the row sums of the obtained incomplete LU decomposition are the same as the row sums of the original matrix,

$$A\mathbf{1}_n = LU\mathbf{1}_n.$$

This method is referred to as Modified ILU or MILU.

5.4.3 ILUT

Factorisations relying on levels of fill-in, like the ILU(p) strategies, in general can not tell anything about the locations in which fill-in happens. In some special cases, like matrices consisting of diagonals, something can be established about the locations of fill-in, but there is no way to generalize this. This means that it is possible for there to be much more fill-in in one part of the matrix than in other parts. Another problem with factorisations like ILU(p) is that the level of fill-in does not tell anything about the magnitude of the fill-in. Because of this a lot of fill-in elements of very low magnitude can be added. The improvement obtained from using low magnitude elements is of low magnitude as well, but the low magnitude elements require the same amount of storage and computation as higher magnitude elements. It might thus be advantageous to remove the low-magnitude elements from the decomposition.

The ILUT, or Incomplete LU with Threshold(s), method uses ways to improve upon these things. Like the ILU(0) and ILU(p) methods observed before, ILUT makes use of Gaussian elimination, with certain rules to drop elements, to compute an LU decomposition of a matrix. However, unlike the previously discussed methods, ILUT does not use a predefined non-zero structure. It instead uses predefined dropping rules to dynamically obtain a non-zero structure.

A common way of applying ILUT is by defining two parameters, a maximum number of elements per row p and a relative tolerance τ, which are used in the Gaussian elimination process used to determine a regular LU-decomposition (Algorithm 5.1 with $P = \emptyset$). The relative tolerance is used to drop updates of small magnitude in the Gaussian elimination process. For any computation to update a value of A, the new value is compared against the relative tolerance multiplied by the norm of the row of the element and the update is only

performed if it is greater than this tolerance. If the updated element is smaller, it is dropped and replaced by 0. The maximum number of elements per row p is used after calculating all elements of the LU decomposition. For both the obtained lower triangular matrix L and upper triangular matrix U only the p largest elements in each row are kept. All other elements are dropped and replaced by 0. In this way, both the amount of elements per row and the minimal magnitude of elements are limited.

Smaller modifications to this process can be made, like the MILU modification described for the earlier discussed ILU methods. This same modification can be applied to ILUT in the same way, by subtracting remainders obtained in the ILU decomposition from the diagonal elements of the obtained matrix U. Another modification that can be used is not dropping the elements in the upper triangle of the matrix that are of largest modulus in their row in the original matrix. So for each row i, there is a position (i, j_l^i), for which $[A]_{i,j_l^i}$ is the largest element of row i. Then the dropping rules are at no point in the Gaussian elimination process applied to elements in position (i, j_l^i). An important result of the combination of these two modifications is that, if the original matrix A is diagonal dominant, then the product of the obtained ILU decomposition, LU, is also diagonal dominant.

5.4.4 ILUS

In certain applications it is common for sparse matrices to be stored in a sparse skyline format. This format stores the diagonal elements and the parts of columns above the diagonal and the parts of rows below the diagonal both as sparse vectors. If a matrix is symmetric, only the vector representing the upper part of columns is required to be stored.

An efficient way of computing an LDU-factorisation of a matrix can be defined specifically for matrices that are stored in this sparse skyline format. To this method dropping rules, similar to those in ILUT can be applied, to obtain an ILU decomposition. This is called the ILUS method.

Suppose that for a matrix $A_k \in \mathbb{R}^{k\times k}$ an LDU-factorisation has been obtained, so

$$A_k = L_k D_k U_k,$$

and using A_k a larger matrix $A_{k+1} \in \mathbb{R}^{(k+1)\times(k+1)}$ can be defined by

$$A_{k+1} = \begin{bmatrix} A_k & \mathbf{v}_k \\ \mathbf{w}_k & \alpha_{k+1} \end{bmatrix}.$$

Then an LDU-factorisation can be obtained as

$$A_k = \begin{bmatrix} L_k & \mathbf{0}_k \\ \mathbf{y}_k & 1 \end{bmatrix} \begin{bmatrix} D_k & \mathbf{0}_k \\ \mathbf{0}_k^T & d_{k+1} \end{bmatrix} \begin{bmatrix} U_k & \mathbf{z}_k \\ \mathbf{0}_k^T & 1 \end{bmatrix},$$

with

$$\begin{aligned} \mathbf{z}_k &= D_k^{-1} L_k^{-1} \mathbf{v}_k \\ \mathbf{y}_k &= \mathbf{w} D_k^{-1} L_k^{-1} \\ d_{k+1} &= \alpha_{k+1} - \mathbf{y}_k D_k \mathbf{z}_k. \end{aligned}$$

To the computation of $\mathbf{z}_k$ and $\mathbf{y}_k$ a dropping strategy can be applied. For example dropping elements if they are smaller than the relative threshold τ and only using the largest p values of $\mathbf{z}_k$ and $\mathbf{y}_k$, like it was done for ILUT. This gives the ILUS(τ, p) method. Applying this method to a symmetric matrix A results in an incomplete Cholevski matrix, which means that symmetries are preserved.

5.5 Approximate Inverse

A downside of the Incomplete LU factorisations discussed in the previous section is that they only work well for specific types of matrices. If the matrix that is required to be preconditioned is diagonally dominant and an M-matrix, then ILU preconditioning works very well. However, if this is not the case the performance of ILU preconditioners can not be guaranteed. It can for example be seen that when A is not diagonally dominant, it can not be guaranteed that LU is well conditioned. For the error matrix, obtained from

$$E = A - LU,$$

the preconditioned error matrix can be computed as

$$L^{-1}EU^{-1} = L^{-1}AU^{-1} - I.$$

Here it is observed that poorly conditioned matrices L and U lead to a preconditioned matrix which may not remain within desired limits.

To deal with these problems, Approximate Inverse preconditioners try to define a preconditioning matrix M which is a direct approximation of the inverse of A and does not require solving a linear system. This is done by finding a matrix that minimizes the Frobenius norm of the residual matrix, for the right preconditioner given by

$$F(M) = \|I - AM\|_F^2 = \sum_{j=1}^{n} \|\mathbf{e}_j - A[M]_{:,j}, \tag{5.12}$$

with similar definitions for the left and split preconditioners.

To obtain the approximation of the inverse, M is treated as an unknown sparse matrix. Then a descent technique that minimizes Equation (5.12) is applied, using M as an unknown in $\mathbb{R}^{n^2}$. Such a technique is defined by iterations of the form

$$M_{i+1} = M_i + \alpha G,$$

where G is a chosen direction and α is such that $F(M_{i+1})$ is minimized. For Equation (5.12) this is done by setting

$$\alpha = \frac{\mathrm{tr}(R^T AG)}{\mathrm{tr}((AG)^T AG)},$$

where $R = I - AM$. Defining a search direction G can be more challenging. A simple way of defining a search direction is by using the previous residual $G = I - AM_i$. Another commonly used method to obtain a search direction is by using the direction of steepest descent, obtained from the gradient of F. This gradient is obtained as $G = -2A^T(I - AM)$. Using the direction steepest descent can lead to very slow convergence and thus might not be desired.

An alternative way of obtaining an approximate inverse is by minimizing the norm columns of the residuals separately. Then n functions of the form

$$f_j(\mathbf{m}) = \|\mathbf{e}_j - A\mathbf{m}_j\|_2^2$$

need to be minimized and matrix M is defined by setting the vectors $\mathbf{m}_j$ as its columns. Minimizing these functions is done by solving the equations of the form $A\mathbf{m}_j = \mathbf{e}_j$, which is again a matrix-vector system. Solving this can be done iteratively, but then it is required to use a small amount of iterations compared to the amount of iterations applied later to the preconditioned original system in Equation (5.2). This is to ensure that the computational work in the process of determining the preconditioner is not more than the computational work required to solve the original system.

6 Conclusion

The objective of the project, for which this literature study was performed, is to investigate the potential of improving the performance of the linear solver used in the Visage finite element geomechanics simulator. This literature study gives a summary of options for solving linear systems, some of which might be useful for improving the linear solver used by Visage.

This report discusses various methods for solving linear systems derived from an FEM discretization of a system of PDEs describing forces and displacements in reservoirs for CCS. For this, it is discussed what such a linear system actually looks like. After this, a wide range of methods for solving these linear systems have been discussed. First, it was established which direct methods for solving such equations can be applied. These methods are able to find accurate solutions to the problems that are studied, but are computationally and memory-wise very costly for large systems. Therefore, it was required to shift to iterative methods, which do not necessarily find an exact solution to a system but are able to make a good approximation of the exact solution and require much less computations and memory than direct methods. Several techniques are discussed to improve these iterative methods, such as multigrid methods and preconditioning.

As this study only discusses possible methods for solving linear systems, it is challenging to draw any substantial conclusions. An important conclusion that can be drawn is that GMRES is unlikely to perform better than methods such as CG. This is because CG is developed specifically for SPD systems, which the systems observed are, and is expected to perform better on these types of systems.

For the upcoming phase of this research, a wide range of questions remains, most importantly involving integrating an implementation of the linear solvers seamlessly with the existing implementation of the Visage simulator. A summary of these questions is given in Table 6.1.

Category	Questions
Current Situation	What are the methods that are currently used to solve linear systems?
	What does the implementation of the currently used solver look like?
Advantage of discussed iterative methods	Which of the methods discussed here can be expected to give an advantage over the current implementation?
	How can these methods be implemented in a way that corresponds the implementation of the currently used solver?
Software Packages for iterative solvers	What packages can be used for solving linear systems using iterative methods?
	How can these packages be used with the Visage software?
Identification of test cases	What are more concrete examples of test cases these solvers are applied to?
Performance of solver	What improvements in performance are observed when applying these solvers to test cases?
	How much can the solver be improved in terms of saved time and memory, while keeping the same accuracy?
Other	What methods, not discussed yet, are available to further improve the solver?

Table 6.1: Questions to be answered during next phase of research

Bibliography

[1] Ronaldo I Borja. "A finite element model for strain localization analysis of strongly discontinuous fields based on standard galerkin approximation". In: *Computer Methods in Applied Mechanics and Engineering* 190.11-12 (2000), pp. 1529–1549.

[2] J.W. Demmel. *Applied Numerical Linear Algebra*. Other Titles in Applied Mathematics. Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104), 1997. ISBN: 9781611971446. URL: https://books.google.nl/books?id=P3bPAgAAQBAJ.

[3] Good Plant Design. "Operation for Onshore Carbon Capture Installations and Onshore Pipelines". In: *Energy Institute: London* (2010).

[4] John B Fraleigh and A Bauregard Raymond. "Linear Algebra 3rd". In: *Reading: Addison Wesley* (1995).

[5] Honglei Liu, Wenhao Shi, and Tianhong Yang. "Numerical modeling on anisotropy of seepage and stress fields of stratified rock slope". In: *Mathematical Problems in Engineering* 2020 (2020), pp. 1–10.

[6] Bert Metz et al. "Carbon dioxide capture and storage. Summary for policymakers". In: (2005).

[7] JCM Pires et al. "Recent developments on carbon capture and storage: An overview". In: *Chemical engineering research and design* 89.9 (2011), pp. 1446–1460.

[8] Yousef Saad. *Iterative methods for sparse linear systems*. SIAM, 2003.

[9] Schlumberger. *Visage Finite-element geomechanics simulator*. Schlumberger Oilfield UK Limited. Abingdon, United Kingdom, 2023.

[10] SLB. *Petrel geomechanics modeling*. 2024. URL: https://www.software.slb.com/products/petrel/petrel-geomechanics/reservoir-geomechanics (visited on 02/14/2024).

[11] SLB. *Visage finite element geomechanics simulater*. 2024. URL: https://www.slb.com/products-and-services/delivering-digital-at-scale/software/visage (visited on 02/14/2024).

[12] Vuik, C. and Lahaye, D.J.P. "Scientific Computing (wi4201) Lecture Notes". 2019.