NTNU

Norwegian University of
Science and Technology

Multigrid preconditioning of the linear elasticity equation

Magnus Jorstad

Master of Science in Physics and Mathematics
Submission date: June 2016
Supervisor: Knut Andreas Lie, MATH
Co-supervisor: Trond Kvamsdal, MATH

Norwegian University of Science and Technology
Department of Mathematical Sciences

Abstract

Multigrid methods are known to be very efficient linear solvers for 2nd order elliptic PDEs. In this thesis we consider multigrid methods, and the use of such as preconditioners, when solving the linear elasticity equation. An introduction to modeling elasticity is given and two discretization techniques; the finite-element and the virtual-element method are presented. The conjugate gradient method is described and a result relating the convergence rate to the condition number is established. Building blocks of the multigrid method are presented. As a part of the study, an implmentation of the smoothed aggregation algebraic multigrid method due to Vanek et al. has been done for elasticity. Numerical tests on simple problems shows that the convergence rate of our implmentation has a moderate dependence on the problem size when used as a standalone solver. When used as a preconditioner for conjugate gradient the convergence rate is found to be practically independent of the problem size compared to standard preconditioners.

Sammendrag

Flergittermetoder er kjent for å være svært effektive lineære løsere for annenordens elliptiske PDEer. I denne masteroppgaven betrakter vi flergittermetoder, og bruk av slike som prekondisjoneringer, for å løse den lineære elastisitetsligningen. En introduksjon til modellering av elastisitet blir gitt og to forskjellige diskretiseringsteknikker; endelig-element metoden og virituell-element metoden, blir presentert. Konjugate gradienters metode blir beskrevet og sammenhengen mellom konvergensraten og kondisjonsnummeret blir utledet. Byggesteiner for flergittermetoder blir presentert. Som en del av arbeidet har vi implementert en algebraisk flergittermetode basert på utjevna aggregater introdusert av Vanek et al. Numeriske eksperimenter på enkle test-problemer viser at konvergensraten for vår implementasjon er svakt avhengig av størrelsen på problemet når metoden benyttes som en frittstående iterativ løser. Når metoden benyttes for å prekondisjonere konjugate gradienters metode viser den seg å være så godt som uavhengig av problemstørrelsen sammenlignet med standard prekondisjoneringer.

Preface

This thesis marks the end of my Master of Science degree in Physics and Mathematics, with specialization in Industrial Mathematics, at the Norwegian University of Science and Technology. The work have been done in connection with the Matlab Reservoir Simulation Toolbox (MRST) developed by SINTEF Applied Mathematics. I would like to thank my advisor Professor Knut-Andreas Lie and co-advisor Professor Trond Kvamsdal for always being available with quick answers and valuable feedback, and for welcoming me at SINTEF Applied Mathematics' office. I would also like to thank Halvor Møll Nilsen, senior scientist at SINTEF Applied Mathematics for sharing MATLAB code used in this project and for discussing topics related to this work. In addition I would like to thank Dr. Arne Morten Kvarving at SINTEF ICT for answering some of the questions I had during my study.

Magnus Jorstad
Trondheim, June 2016

Contents

Chapter 1

Introduction

Reservoir simulations have been an active research area for decades. The search for better, more accurate simulation methods has given rise to the construction of more involved mathematical models for describing physical situations, both in terms of structure and problem size. In particular, coupling of different physical phenomena is an increasing trend and can improve quality of simulations drastically and give the ability to simulate increasingly complex situations. Linear solvers are a cornerstone for most numerical simulations. More complex mathematical models requires more of the linear solvers.
Hydraulic fracturing is a technique of enhancing flow of hydrocarbons or water in reservoirs, by injecting high-pressurized fluids in wells. Simulating this, geomechanics plays an important role. Typical for reservoir simulations is that they are performed on large models and the material properties are often highly complex. This leads to difficulties when solving the linear systems.

In this thesis the object is to investigate iterative linear solvers applicable to systems arising in geomechanics. To do this we need to have knowledge about the origin of the linear system. For mechanics in general, the finite-element method is the most widely used discretization method. When dealing with reservoirs, the grids are often highly complex, which makes the finite-element unsuited. Other discretization methods have been proposed, which are better suited for such grids. We mention the multi-point stress approximation (MPSA) [38], which generalize to elasticity well established methods for reservoir flow simulations. We consider here, instead, the recently developed virtual element method [5, 14, 19] which has a closer resemblance to the finite-element framework. As for finite-element, this leads to a symmetric positive definite system.
The conjugate gradient has proven to be a very efficient iterative framework for symmetric positive definite problems. However, for complex problems, it relies on good preconditioners, or "approximate solvers", in order to retain its efficiency. In the search for preconditioners that performs well for large systems, multigrid methods are a natural choice, having a theoretical convergence rate independent of the problem size. In order to design efficient multigrid methods for the elasticity equation we need to understand the properties of the linear system. A goal of the work done in this thesis is to understand how these properties affect the process of solving the linear system iteratively, and in particular using multigrid methods.

1.1 Structure of thesis

In Chapter 2 we start by giving an introduction to the mathematical modeling of elasticity, where governing equations and methods of describing the elastic property of materials are presented. The finite-element method, being the standard discretization method, will be presented, followed by the virtual-element method, extending the properties of the finite-element method to more general grids. In Chapter 3 we

consider iterative linear solvers, describing the conjugate gradient method and stationary iterative methods, which will serve as important building blocks for multigrid methods, presented in Chapter 4. In Chapter 5, a MATLAB implementation of a specific multigrid method is described. In Chapter 6 some numerical case-studies illustrating the concepts presented in this thesis are done, before we conclude the work in Chapter 7.

1.2 Notation

We try to follow relatively standard notation throughout the thesis. In Chapter 2, where we describe continuous quantities in 3D space, such as position and force, we make use of a bold font for continuous quantities with more than one cmoponents, e.g. $\boldsymbol{u} = (u_1, u_2, u_3)$. Later, when describing discrete vectors in higher dimensions we adapt to conventional notation, writing vectors and matrices without bold font. Throughout the thesis d denotes the spatial dimension.

Chapter 2

Reservoir mechanics

2.1 Modeling elasticity

R^{3} -> R^{3}

To model the deformation of a three dimensional body we introduce a displacement field $\boldsymbol{u} = \boldsymbol{u}(\boldsymbol{x})$. This is a vector field defining the displacement from the equilibrium state in each of the three directions as a function of the position $\boldsymbol{x} = (x, y, z)$. Another fundamental quantity in modeling mechanics is the stress field $\boldsymbol{\sigma} = \boldsymbol{\sigma}(\boldsymbol{x})$. This tells us what the axial and shear stresses are at every point in the body. The stress $\boldsymbol{\sigma}$ is a second order tensor, which can be represented with a total of nine elements where three correspond to normal stresses and six to shear stresses,

$$\boldsymbol{\sigma} = \begin{pmatrix} \sigma_{xx} & \sigma_{xy} & \sigma_{xz} \\ \sigma_{yx} & \sigma_{yy} & \sigma_{yz} \\ \sigma_{zx} & \sigma_{zy} & \sigma_{zz} \end{pmatrix}. \tag{2.1}$$

An illustration of these nine quantities is shown in Figure 2.1

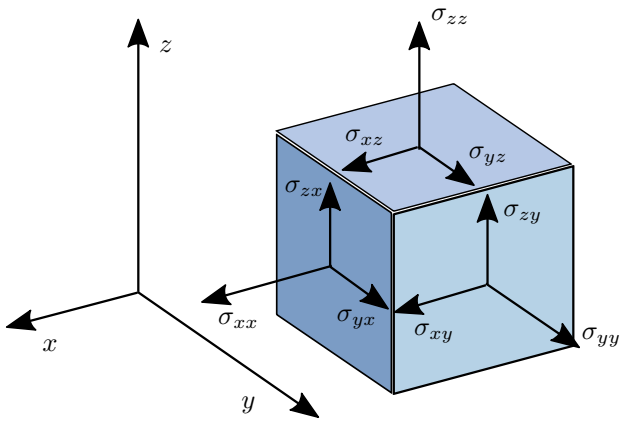


Figure 2.1: The nine quantities in the stress tensor $\boldsymbol{\sigma}$ on a representative volume.

Under the assumption that we have reached a static equilibrium we have that the net angular momentum of some region in the body is zero. This implies that $\sigma_{ij} = \sigma_{ji}$, which means that $\boldsymbol{\sigma}$ is a symmetric tensor. We thus write $\boldsymbol{\sigma} \in \mathbb{S} = \{A \in \mathbb{M}^{d\times d} : A \text{ symmetric}\}$. Another important field in deformation problems is the strain $\boldsymbol{\varepsilon}$. This is a dimensionless representation of the deformation at a given point. As for the stress field the strain can be represented as a second order tensor $\boldsymbol{\varepsilon} \in \mathbb{S}$. For a given displacement $\boldsymbol{u}$, define the strain as the symmetric gradient of the displacement state, i.e.,

$$\boldsymbol{\varepsilon}(\boldsymbol{u}) = \frac{1}{2}(\nabla \boldsymbol{u} + \nabla^T \boldsymbol{u}). \tag{2.2}$$

9*9

In general the stress is related to the strain through a nonlinear function, but for sufficiently small deformations, and assuming the material in question has an elastic behavior, the stress-strain relation can be approximated well using a linear relationship. This relationship, called Hooke's law, defines the constitutive equation for linear elastic theory and is given by

$$\boldsymbol{\sigma}(\boldsymbol{u}) = C\boldsymbol{\varepsilon}(\boldsymbol{u}). \tag{2.3}$$

Here, C is the stiffness tensor. This is the physical parameter describing the elastic behavior of the material and is a symmetric positive definite fourth-order tensor. The structure and different values of this tensor are discussed below. To model the mechanical behavior of a body, one needs, in addition to the constitutive law (2.3), a set of balancing equations. This is simply found by applying Newtons law to an arbitrary region in the body and yields

$$-\operatorname{Div}(\boldsymbol{\sigma}) = \boldsymbol{f}_B. \tag{2.4}$$

The notation $\operatorname{Div}(\boldsymbol{\sigma})$ represents the divergence operator applied row-wise to the stress tensor. The right hand side of (2.4) is the body force, which typically just represents gravity.
To have a well-posed model we need to specify some boundary conditions as well. These can either be a given displacement on the boundary, which we call a pure displacement problem, some specified traction on the boundary, which is called a pure traction problem, or a combination of those. In general any part of the body can be given such conditions, but to have a well-posed problem certain requirements have to be met. These are discussed further in Section 2.3.

2.2 Elastic anisotropy

The layered structure in reservoirs tends to give an anisotropic elastic behavior. This means that the deformation under a given load depends on which direction the load is applied. This elastic behavior is reflected in the stiffness tensor C. As C is a fourth-order tensor it is useful to introduce Voigt's notation which is a way of representing a symmetric tensor by reducing its order. Because of the symmetry property of the stress and strain tensor, there are only six independent elements in each of them, three corresponding to the axial deformation and three to shear deformation. Therefore each of these fields can be represented as a vector with six elements. Using the conventional ordering of the six elements, the stress and strain vectors are defined as

$$\begin{aligned}\left(\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5, \sigma_6\right) &\coloneqq \left(\sigma_{xx}, \sigma_{yy}, \sigma_{zz}, \sigma_{yz}, \sigma_{xz}, \sigma_{xy}\right), \\ \left(\varepsilon_1, \varepsilon_2, \varepsilon_3, \varepsilon_4, \varepsilon_5, \varepsilon_6\right) &\coloneqq \left(\varepsilon_{xx}, \varepsilon_{yy}, \varepsilon_{zz}, 2\varepsilon_{yz}, 2\varepsilon_{xz}, 2\varepsilon_{xy}\right).\end{aligned} \tag{2.5}$$

Note that the new shear strains are defined as the sum of the two symmetric components, which appears as a factor of two in (2.5). The reason for this is to preserve the scalar invariance, i.e., $\boldsymbol{\sigma} \cdot \boldsymbol{\varepsilon}$ in this new notation should equal the product using the matrix notation (2.1). Likewise, the stiffness tensor C can be represented as a 6×6 symmetric matrix. We then rewrite Hooke's law (2.3) as

$$\begin{pmatrix}\sigma_1\\ \sigma_2\\ \sigma_3\\ \sigma_4\\ \sigma_5\\ \sigma_6\end{pmatrix} = \begin{pmatrix}C_{11} & C_{12} & C_{13} & C_{14} & C_{15} & C_{16}\\ & C_{22} & C_{23} & C_{24} & C_{25} & C_{26}\\ & & C_{33} & C_{34} & C_{35} & C_{36}\\ & & & C_{44} & C_{45} & C_{46}\\ & sym & & & C_{55} & C_{56}\\ & & & & & C_{66}\end{pmatrix} \begin{pmatrix}\varepsilon_1\\ \varepsilon_2\\ \varepsilon_3\\ \varepsilon_4\\ \varepsilon_5\\ \varepsilon_6\end{pmatrix}. \tag{2.6}$$

For a 3D material there are different degrees of anisotropy. These are determined by the existence of material symmetry planes and rotation axes. For a completely isotropic material where no symmetries are present the stiffness tensor contain 21 independent values as in (2.6).
On the other end of the scale, for a completely isotropic material, the stiffness tensor can be represented with only two independent quantities. For this case the stiffness tensor can be written

$$C = \frac{E}{(1+\nu)(1-2\nu)} \begin{pmatrix} 1-\nu & \nu & \nu & 0 & 0 & 0 \\ & 1-\nu & \nu & 0 & 0 & 0 \\ & & 1-\nu & 0 & 0 & 0 \\ & & & (1-2\nu)/2 & 0 & 0 \\ & sym & & & (1-2\nu)/2 & 0 \\ & & & & & (1-2\nu)/2 \end{pmatrix}, \tag{2.7}$$

where E is Young's modulus and is defined as the ratio of the stress along an axis to the strain along the same axis. Poisson's ratio ν relates the stress along one axis to the strain along the two other axes. For example under compression in one direction most materials tends to expand in the two other. This effect is characterized by a positive ν. For a completely incompressible material Poisson's ratio is 0.5 which is the larges it can be. Typical values of ν for reservoir rock are in the range 0.2-0.45, [22]. For the isotropic material Hooke's law (2.3) can be written in a compact form, using the tensor notation of stress and strain, as

$$\boldsymbol{\sigma} = 2\mu\boldsymbol{\varepsilon} + \lambda \mathrm{tr}(\boldsymbol{\varepsilon})I, \tag{2.8}$$

where $\mathrm{tr}(\boldsymbol{\varepsilon})$ denotes the trace of $\boldsymbol{\varepsilon}$. The material parameters μ and λ are known as the Lamé constants and represent resistance to shear deformation and compression, respectively. These constants relate to Young's modulus and Poisson's ratio as

$$\mu = \frac{E}{2(1+\nu)} \tag{2.9}$$

and

$$\lambda = \frac{E\nu}{(1+\nu)(1-2\nu)}. \tag{2.10}$$

In many cases there are three orthogonal symmetry planes, which without loss of generality can be assumed to be the coordinate planes. This essentially means that under shear strain no axial stresses are induced, and that the three shear stresses are decoupled from each other. Such a material is called orthotropic and will have a stiffness tensor of the following structure in Voigt's notation.

$$C^{\mathrm{ort}} = \begin{pmatrix} C_{11} & C_{12} & C_{13} & 0 & 0 & 0 \\ & C_{22} & C_{23} & 0 & 0 & 0 \\ & & C_{33} & 0 & 0 & 0 \\ & & & C_{44} & 0 & 0 \\ & sym & & & C_{55} & 0 \\ & & & & & C_{66} \end{pmatrix} \tag{2.11}$$

For the sedimentary rocks that often are found in a reservoir another simpler model can be looked at which describes a layered structure. Consider a volume built up of multiple thin layers, each consisting of an isotropic material, for example a grid cell much larger then the layer thickness. To look at how this volume behaves it is convenient to make a homogenized model where the material is considered homogeneous inside the volume with constant material properties. Such a process is called upscaling and is an important aspect of reservoir mechanics [11]. Under this homogenization the isotropy in each layer will, when the layers are different, sum up to an anisotropic behavior for the whole volume. An illustration of this is shown in Figure 2.2 where a cube consisting of two different isotropic materials in layers, undergoes compression in two different directions due to a pressure of equal magnitude. When the

pressure is applied to the sides parallel with the layers (in the middle of Figure 2.2), most deformation occurs in the weaker layers while the stiffer layers are not affected as much. When the pressure is applied in the other direction (to the right in Figure 2.2), the weaker material is not free to deform as a single isotropic material, but is restricted by the stiffer layers (assuming a no-sliding condition at the layer interfaces). Looking at the cube as a whole one can see that the total strain as a response of the pressure is larger when it is applied to the sides parallel with the layers. This indicates that the homogenized cube should have the anisotropic property $C_{33} \ll C_{11}$. Because of the isotropy in each layer the elastic behavior is also invariant under rotation around the axis perpendicular to the layers which means that $C_{11} = C_{22}$, $C_{13} = C_{23}$ and $C_{44} = C_{55}$. Such a material is called transversely isotropic. This is a subgroup of the orthotropic materials so the stiffness tensor inherits the structure in (2.11), and can now be written as

$$C^{\text{tri}} = \begin{pmatrix} C_{11} & C_{12} & C_{13} & 0 & 0 & 0 \\ & C_{11} & C_{13} & 0 & 0 & 0 \\ & & C_{33} & 0 & 0 & 0 \\ & & & C_{44} & 0 & 0 \\ & sym & & & C_{44} & 0 \\ & & & & & C_{66} \end{pmatrix}. \tag{2.12}$$

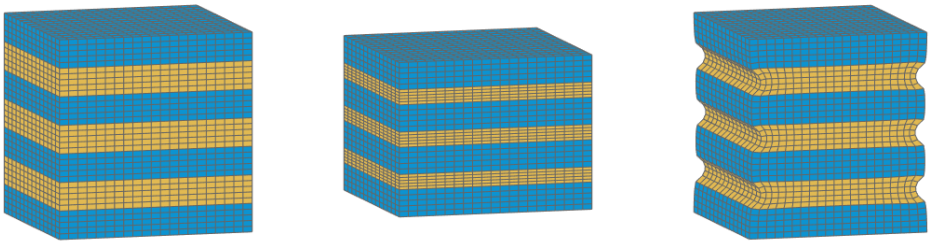


Figure 2.2: A layered material where the layers are parallel and made of two different isotropic materials, under applied stress in different directions. The color indicates different values of Young's modulus, where one is ten times the other. To the left no forces or displacement boundary conditions are applied. In the middle a pressure p is applied at the top and bottom, while to the right p is applied on the left and right side. Figure produced using MRST [17].

2.3 Finite-element method

The most conventional discretization technique for the elasticity equation is the finite-element method. This is widely used and has a large, mature literature built around it. A basic concept of the finite-element method is that the governing equation is looked at in a weak sense. This means that instead of only looking for solutions that explicitly satisfy the given equations, we look for solutions that satisfy the equation only with respect to some given test functions, assuming a proper pairing of two functions can be defined. There are many types of weak formulations of the elasticity problem, for a survey see [18]. We consider here the primal formulation.

For the elasticity problem we are essentially looking at three different fields of unknowns; the displacement $\boldsymbol{u}$, the stress $\boldsymbol{\sigma}$ and the strain $\boldsymbol{\varepsilon}$. The displacement is linked to the strain through the kinematic equation (2.2) and strain to stress through the constitutive equation (2.3). The problem statement is closed using the balancing equation (2.4) and applying displacement boundary conditions and traction boundary conditions.

To be able to formulate the problem weakly it is necessary to define the appropriate functions and spaces. Let us start by considering an linearly elastic body occupying the closed and bounded region $\Omega \in \mathbb{R}^d$,

whose boundary is denoted $\partial\Omega$. Inside Ω, the body is subject to some body forces $\boldsymbol{f}_B \in L^2(\Omega, \mathbb{R}^d)$. Let us for simplicity only consider a pure displacement problem with zero boundary condition, i.e., $\boldsymbol{u} = 0$ on $\partial\Omega$. For a weak formulation of this problem we consider functions in the first Sobolev space satisfying the zero boundary condition, denoted by $H_0^1(\Omega, \mathbb{R}^d)) = \{\boldsymbol{u} \in H^1(\Omega, \mathbb{R}^d) : \boldsymbol{u} = 0 \text{ on } \partial\Omega\}$.

For the primal formulation the constitutive equation (2.3) is considered as a strong relation, and inserted into the balancing equation (2.4), eliminating $\boldsymbol{\sigma}$ as an unknown. The problem can then be compactly written

$$\begin{aligned} -\text{Div } C\boldsymbol{\varepsilon}(\boldsymbol{u}) &= \boldsymbol{f}_B \quad \text{in } \Omega, \\ \boldsymbol{u} &= 0 \quad \text{on } \partial\Omega. \end{aligned} \tag{2.13}$$

Using Riesz Representation Theorem [8] this can be uniquely written

$$-\int_\Omega \text{Div } C\boldsymbol{\varepsilon}(\boldsymbol{u}) \cdot \boldsymbol{v} \mathrm{d}\boldsymbol{x} = \int_\Omega \boldsymbol{f}_B \cdot \boldsymbol{v} \mathrm{d}\boldsymbol{x}, \tag{2.14}$$

for any *test-function* $\boldsymbol{v} \in H_0^1(\Omega, \mathbb{R}^d)$. Applying Green's first identity for the integral on the left hand side we get

$$\int_\Omega C\boldsymbol{\varepsilon}(\boldsymbol{u}) : \boldsymbol{\varepsilon}(\boldsymbol{v}) \mathrm{d}\boldsymbol{x} = \int_\Omega \boldsymbol{f}_B \cdot \boldsymbol{v} \mathrm{d}\boldsymbol{x} + \int_{\partial\Omega} (C\boldsymbol{\varepsilon}(\boldsymbol{u})\boldsymbol{n}) \cdot \boldsymbol{v} \mathrm{d}\boldsymbol{x} \tag{2.15}$$

=0, since assume no traction

where $C\boldsymbol{\varepsilon} : \boldsymbol{\varepsilon} = \Sigma_{i,j=1}^n (C\varepsilon)_{ij}\varepsilon_{ij}$ and $\boldsymbol{n}$ is the outward normal of $\partial\Omega$. As we assume a pure displacement problem and consider only test functions in $H_0^1(\Omega, \mathbb{R}^d)$ the boundary integral on the right hand side disappears. To write the problem more compactly we introduce the notation of the symmetric bilinear form

$$a(\boldsymbol{u}, \boldsymbol{v}) = \int_\Omega C\boldsymbol{\varepsilon}(\boldsymbol{u}) : \boldsymbol{\varepsilon}(\boldsymbol{u}) \mathrm{d}\boldsymbol{x}, \tag{2.16}$$

which is defined for any two elements in $H^1(\Omega, \mathbb{R}^d)$. We can then write the following primal formulation of the linear elasticity equation: Find $\boldsymbol{u} \in H_0^1(\Omega, \mathbb{R}^d)$ such that

$$a(\boldsymbol{u}, \boldsymbol{v}) = \langle \boldsymbol{f}_B, \boldsymbol{v} \rangle, \quad \forall\, \boldsymbol{v} \in H_0^1(\Omega, \mathbb{R}^d), \tag{2.17}$$

where $\langle \cdot, \cdot \rangle$ is the L^2 - inner product. The existence and uniqueness of (2.17) is established in [3].

Another approach that is commonly used in mechanics is based on the principle of virtual work [51]. Here the solution is found as the minimizer of the total potential energy of the system. The strain energy density function is given as

$$W(\boldsymbol{u}) = \frac{1}{2}\, \boldsymbol{\sigma}(\boldsymbol{u}) : \boldsymbol{\varepsilon}(\boldsymbol{u}) = \frac{1}{2}\, C\boldsymbol{\varepsilon}(\boldsymbol{u}) : \boldsymbol{\varepsilon}(\boldsymbol{u}). \tag{2.18}$$

The total potential energy is found by integrating $W(\boldsymbol{u})$ over the whole domain Ω and subtract the work done by the applied force. The solution of (2.13) is found as the minimizer of

$$\min_{\boldsymbol{u} \in H_0^1(\Omega, \mathbb{R}^d)} \quad \frac{1}{2} \int_\Omega C\boldsymbol{\varepsilon}(\boldsymbol{u}) : \boldsymbol{\varepsilon}(\boldsymbol{u}) \mathrm{d}\boldsymbol{x} - \int_\Omega \boldsymbol{f}_B \boldsymbol{u} \mathrm{d}\boldsymbol{x}. \tag{2.19}$$

Under the assumption that C is symmetric positive definite the two formulations above are equivalent because the necessary and sufficient condition for the minimizer of (2.19) is exactly that (2.17) must hold for every test function $\boldsymbol{v} \in H_0^1(\Omega, \mathbb{R}^d)$.

2.3.1 Discretization

Let us now find a finite-element approximation of (2.17). To do this we need to find a finite dimensional approximation of $H^1(\Omega, \mathbb{R}^d)$. Let us start by constructing a grid, or a mesh, by defining a partition $\mathcal{T}_h$ of the domain Ω into non-overlapping elements of size h, i.e., for any element $K \in \mathcal{T}_h$ we have $\max_{\boldsymbol{x},\boldsymbol{y}\in K} \|\boldsymbol{x}-\boldsymbol{y}\|_2 \leq h$. Coners of elements will be referred to as nodes and we denote the total number of nodes in $\mathcal{T}_h$ by n. Let us here use a standard finite-element discretization and assume that all elements are simplexes, as in Figure 2.3.

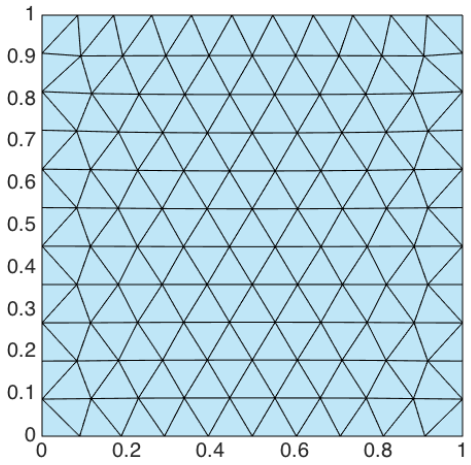


Figure 2.3: Typical finite-element mesh on the unit square. Generated using [40].

Now we can define a set of basis functions on each element that is supposed to approximate the restriction of $H^1(\Omega, \mathbb{R}^d)$ to the element. The simplest and most widely used are the continuous linear nodal basis functions, which on simplexes are uniquely defined by the kronecker delta property on each node $\boldsymbol{n}_j$ of the element,

$$\phi_i(\boldsymbol{n_j}) = \delta_{ij}, \quad i,j = 1,...,(d+1). \tag{2.20}$$

* 张量积

In 2D this gives the three basis functions in Figure 2.4. This basis constitutes a first order approximation

Figure 2.4: Linear nodal basis functions on a triangular element.

of $H^1(\Omega, \mathbb{R}^d)$, which means that the solution $\boldsymbol{u}_h$ found in this space will have an error that is $\mathcal{O}(h)$. A powerful feature of the finite-element method is that one can, in general, increase the approximation order to an arbitrary degree by simply choosing basis elements as polynomials up to the desired degree.

One can notice that the function ϕ_i is written as a scalar function, and not a vector. The generalization to our case where we model a vectorfield is straight-forward, by simply using the same scalar function in each space-variable. On simplexes, the scalar function ϕ_i is equivalent with the uniquely defined barycentric coordinate of node i in an element. Such coordinates can be useful in constructing different finite dimensional spaces and will be discussed more in the section on the virtual-element method.

By requiring continuity across all edges, or faces in 3D, we end up with one degree of freedom per node in the global setting. That is, a global basis function is constructed as the piecewise linear composition of each element basis function that has the value 1 on a given node. The global basis function ϕ_i has support only in the elements that has node i on its boundary. As mentioned above, we are modeling a d-dimensional vector field, which means that we get a total of $N = nd$ basis functions, where n is the number of nodes of $\mathcal{T}_h$. We can now approximate $H^1(\Omega, \mathbb{R}^d)$ by the finite dimensional vector space

$$V = span\{\boldsymbol{\varphi}_1, \boldsymbol{\varphi}_2, ..., \boldsymbol{\varphi}_N\}. \tag{2.21}$$

Having properly defined this space the discrete version of the continuous problem (2.17) can be stated as: Find $\boldsymbol{u}_h \in V$ such that

V = N-dimensional linear space

$$a(\boldsymbol{u}_h, \boldsymbol{v}) = \langle \boldsymbol{f}_B, \boldsymbol{v} \rangle, \quad \forall\ \boldsymbol{v} \in V, \tag{2.22}$$

which is called the Galerkin approximation of (2.17). For this finite dimensional problem we can write the solution $\boldsymbol{u}_h$ as a linear combination of the basis elements,

$$\boldsymbol{u}_h = \sum_{i=1}^{N} \chi_i \boldsymbol{\varphi}_i, \tag{2.23}$$

where the coefficients χ_i will be referred to as the degrees of freedom. As the basis functions span the whole space it is clear that we need only consider each basis element as test functions. Using the linearity of the bilinear form $a(\cdot, \cdot)$ we can write (2.22) as

$$\sum_{i=1}^{N} \chi_i a(\boldsymbol{\varphi}_i, \boldsymbol{\varphi}_j) = \langle \boldsymbol{f}_B, \boldsymbol{\varphi}_j \rangle, \quad \forall\ j = 1, ..., N. \tag{2.24}$$

This notation clearly motivates a matrix-vector equation. We write the linear system resulting from the primal finite-element discretization of the elasticity equation as

$$Ax = F, \tag{2.25}$$

where the system matrix and right hand side are given by

$$A_{ij} = a(\boldsymbol{\varphi}_i, \boldsymbol{\varphi}_j), \qquad F_j = \langle \boldsymbol{f}_B, \boldsymbol{\varphi}_j \rangle \tag{2.26}$$

and $x \in \mathbb{R}^N$ is a vector containing the degrees of freedom $\chi_i,\ i = 1, ..., N$.
To compute the matrix coefficients we can take advantage of the relatively small support of each basis function. In practice it is common to iterate through all the elements and compute element contributions to A and F by considering only i and j corresponding to nodes of the current element. The element contributions are computed as

$$A_{ij}^K = \int_K C\boldsymbol{\varepsilon}(\boldsymbol{\varphi}_i) : \boldsymbol{\varepsilon}(\boldsymbol{\varphi}_j) \mathrm{d}\boldsymbol{x} \qquad \text{and} \qquad F_j^K = \int_K \boldsymbol{f}_B \cdot \boldsymbol{\varphi}_j \mathrm{d}\boldsymbol{x}, \tag{2.27}$$

where i and j are local indexes of the nodes on element K. For each element K one can define a mapping T_K that assigns to each local degree of freedom a global number. While this mapping in practice can be done by direct assignment, we can think of it as a highly sparse, binary matrix whose rows correspond to local degrees of freedom and columns to global. One can then assemble the global matrix element by element as

$$A \ \leftarrow \ A + T_K^T A^K T_K \tag{2.28}$$

and the global force vector as

$$F \leftarrow F + T_K^T F^K. \tag{2.29}$$

In elasticity it is common to use an interleaved ordering of the unknowns, meaning the degree of freedom representing the x-displacement at node i is followed by the degrees of freedom representing the y- and z-displacement at the same node. This results in a natural block structure in A where the size of each block equals the spatial dimension. An example of the sparsity pattern of a global matrix in the case of a 2D triangular grid is shown in Figure 2.5, where the 2-by-2 blocks are seen.

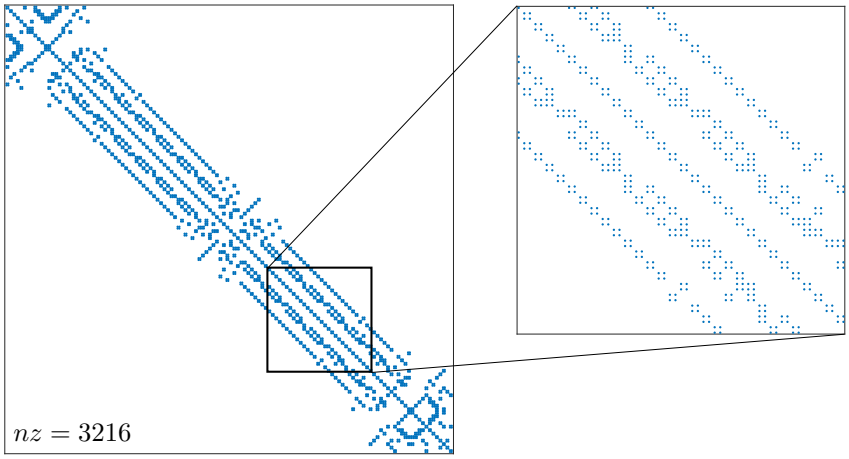


Figure 2.5: Example of sparsity pattern of the system matrix in (2.25). The trianglular grid in Figure 2.3 is used.

One can notice that when using linear basis functions the integrand is constant in the expression for A^K in (2.27). Hence A^K can be computed exactly using only geometric information about the element K. To compute the force vector F^K a suitable quadrature rule must be used to numerically integrate, unless the body force is constant.

2.3.2 Boundary conditions and rigid body modes

In order to find a unique solution to (2.25) we need F to be in the range of A, or better yet, A should be nonsingular. If A is positive definite then this is satisfied for any $F \in \mathbb{R}^N$. To investigate this we consider the bilinear form $a(\cdot,\cdot)$, which is defined in (2.16). The stiffness tensor C is symmetric positive definite. This means that $a(\cdot,\cdot)$ is symmetric in its arguments and for any non-zero matrix $\boldsymbol{\tau}$ the L^2-inner product $\langle C\boldsymbol{\tau}, \boldsymbol{\tau}\rangle$ is positive. The symmetric gradient operator $\boldsymbol{\varepsilon}(\cdot)$, however, has a non-empty kernel. From this it follows that the bilinear form $a(\cdot,\cdot)$ is positive semidefinite, and not strictly positive definite. Let us take a closer look at the kernel of $a(\cdot,\cdot)$, which here is defined as

$$ker(a) = \{\boldsymbol{v} \in H^1(\Omega, \mathbb{R}^d) \ : \ a(\boldsymbol{v}, \boldsymbol{v}) = 0\}. \tag{2.30}$$

Because of the definiteness of C we can conclude that $ker(a) = ker(\boldsymbol{\varepsilon})$. It is clear that constant functions, whose derivative is zero, are in $ker(a)$. In our case these constant kernel elements correspond to translation modes, which indeed have zero strain energy associated with them. The number of independent translation modes equals the spatial dimension and can be represented by the canonical basis

$$\boldsymbol{t}_1(\boldsymbol{x}) = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \qquad \boldsymbol{t}_2(\boldsymbol{x}) = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \qquad \boldsymbol{t}_3(\boldsymbol{x}) = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}. \tag{2.31}$$

Because of the symmetric definition of $\boldsymbol{\varepsilon}$, see (2.2), there are in addition to these constant functions some linear functions that result in zero strain. These functions correspond to rotational modes. In 3D these modes are given as

$$\boldsymbol{r}_1(\boldsymbol{x}) = \begin{bmatrix} 0 \\ z \\ -y \end{bmatrix}, \qquad \boldsymbol{r}_2(\boldsymbol{x}) = \begin{bmatrix} -z \\ 0 \\ x \end{bmatrix}, \qquad \boldsymbol{r}_3(\boldsymbol{x}) = \begin{bmatrix} y \\ -x \\ 0 \end{bmatrix}, \tag{2.32}$$

corresponding to rotation around the x- y- and z-axis respectively. Figure 2.6 shows the three rigid body modes in 2D. Notice that while the rotational modes above describe rotation around the coordinate

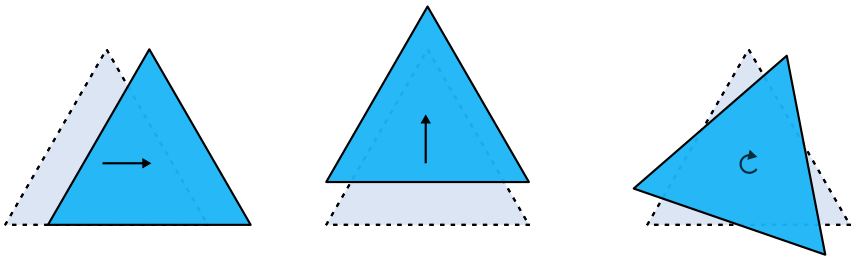


Figure 2.6: Three rigid body modes in 2D.

axes, one can describe rotation around any axis as a linear combination of $\boldsymbol{t}_1, \boldsymbol{t}_2, \boldsymbol{t}_3$ and $\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{r}_3$. It is straight-forward to show that the above set of 6 rigid body modes span the kernel of $\boldsymbol{\varepsilon}$, and hence the kernel of $a(\cdot, \cdot)$. We define the space of rigid body modes

$$\mathcal{R} := ker(a) = span\{\boldsymbol{t}_1, \boldsymbol{t}_2, \boldsymbol{t}_3, \boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{r}_3\}. \tag{2.33}$$

Having investigated the kernel of $a(\cdot, \cdot)$, let us return to the question of whether (2.25) has a unique solution. From a physical point of view one can argue that some part of the body has to be locked to prevent rigid body motions. In the derivation above a zero displacement at all boundary nodes is assumed, which means we require the solution to be in $H_0^1 \subset H^1$. Considering only functions in H_0^1 one can see that all the kernel elements vanish, i.e.,

$$\mathcal{R} \cap H_0^1 = \emptyset, \tag{2.34}$$

and the body is no longer free to float. This means that in the continuous formulation (2.17) the solution is unique. One can notice, however, that in the Galerkin approximation (2.22) we approximate H^1 and not just H_0^1 by including the degrees of freedom at the boundary. Although this larger space is not necessary in our zero boundary displacement case, it is a more general way that allows for other boundary conditions. This gives us a singular system where the eigenvectors with zero eigenvalue are those corresponding to the six rigid body modes above.

The boundary conditions for this system are yet to be imposed. One way to do this is to define a logical map or set of indexes that indicates which of the degrees of freedom that correspond to Dirichlet nodes. Let for example D be the part of the identity matrix that is one only at those diagonal entries that correspond to Dirichlet degrees of freedom. Let $\widetilde{D}$ be the rest of the identity matrix so that we get the partition $I = D + \widetilde{D}$. Using this partition we can separate our vector into an unknown and a known part,

$$x = Dx + \widetilde{D}x \tag{2.35}$$

The known terms can then be moved to the right hand side as

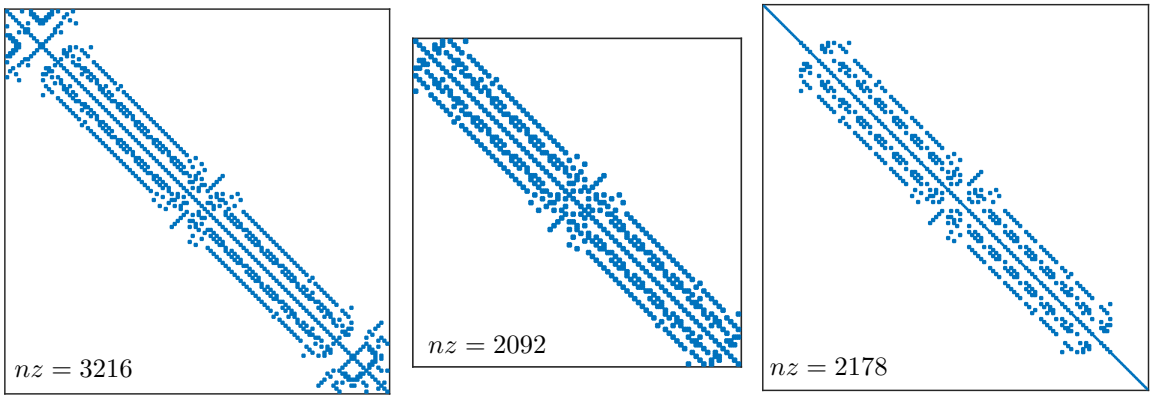
$$ADx = F - A\widetilde{D}x. \tag{2.36}$$

imposing dirichlet boundary !

O(n^2) for multiplying diagonal matrix

This system is generally over-determined as it contains more equations than unknowns. We remove all the equations that correspond to the known values by left-multiplying the equation by $\widetilde{D}$.

$$\widetilde{D}A\widetilde{D}x = \widetilde{D}F - \widetilde{D}ADx. \tag{2.37}$$



(a) Full system A including Dirichlet degrees of freedom

(b) Collapsed system containing only equations between unknowns

(c) Reduced system $\widetilde{D}A\widetilde{D}$ with added diagonal.

Figure 2.7

We can now collapse the matrix, see Figure 2.7b, by discarding the zero rows and columns and obtain a symmetric positive definite system that is invertible. It can, however, be useful to keep the uncollapsed system when designing preconditioners, as will be discussed in section 4.2. We can then instead implicitly set the dirichlet degrees of freedom by adding a diagonal element to the uncollapsed $\widetilde{D}A\widetilde{D}$ and thus recover a nonsingular system as in 2.7c.

2.4 Virtual-element method

As mentioned above the finite-element method has the nice property of polynomial accuracy on simplexes. This means that for a higher accuracy, one simply needs to choose a higher order of the polynomial shape functions. In many practical applications, however, it is desirable to use a grid not consisting of simplexes, but of general polyhedral elements. For such an element it is not possible to define linear basis functions that are continuous and satisfy the Kronecker delta property on all vertices of the element, as we did for the finite-element method above. The recently developed virtual-element method [5, 14, 19] can be viewed as a modification of the finite-element method to extend this nice property to general polyhedra by adding to the finite-element function space some additional functions that satisfy certain properties. For simplexes the two methods coincide.

We use here the same continuous framework as we did for the finite-element method, i.e., we seek a solution $\boldsymbol{u} \in H_0^1(\Omega, \mathbb{R}^d)$ such that

$$a(\boldsymbol{u}, \boldsymbol{v}) = \langle \boldsymbol{f}_B \cdot \boldsymbol{v} \rangle, \quad \forall\ \boldsymbol{v} \in H_0^1(\Omega, \mathbb{R}^d). \tag{2.38}$$

To introduce a discretization of (2.38), we start by defining a partition $\mathcal{T}_h$ of Ω into general polyhedra. On each element $K \in \mathcal{T}_h$ we define a finite dimensional element function space $\mathcal{W}_K$. Let n_K be the number of vertices on K.
We now want to find a set of basis functions for $\mathcal{W}_K$ that are continuous across element boundaries and satisfy the Kronecker delta property on each vertex of K. As mentioned in the previous section

the continuous linear nodal basis functions on simplexes are equivalent to the uniquely defined linear barycentric coordinates. On polyhedra such linear coordinates does not exist. By allowing the coordinates to be non-linear at the interior of K we can construct a set of coordinates $\phi_1, ..., \phi_{n_K}$ that satisfy the Kronecker delta property and are linear on ∂K. This non-linearity opens for a variety of different choices to construct such coordinates [31, 26, 6, 45]. To keep the presentation simple we consider only the 2D case here. For an extension to 3D, see [19].

A simple method to construct ϕ_i in 2D is to use *harmonic lifting*. Then the barycentric coordinate ϕ_i for node i is determined as the solution of the Laplace equation with the Dirichlet boundary condition that it should equal to one at node i, decay linearly on edges incident to vertex i and vanish on the rest of the element boundary ∂K. Using these coordinates as nodal basis functions for both x- and y-displacement we get a set of basis functions that uniquely determines the deformation at the interior of K. Also, since ϕ_i is linear at ∂K we have continuity across element boundaries in the global function space $\mathcal{W}$ which makes it conforming. Another important observation is that if we are to represent a linear function by these nodal basis functions, the values at each node would lie in the same plane, i.e., for some $\boldsymbol{a}, \boldsymbol{b} \in \mathrm{R}^d$,

$$\boldsymbol{u}_h(\boldsymbol{v}_i) = \boldsymbol{a} \cdot \boldsymbol{v}_i + \boldsymbol{b}, \qquad i = 1, ..., n_K. \tag{2.39}$$

Then the value at the interior of K would also lie in this plane,

$$\boldsymbol{u}_h(\boldsymbol{x}) = \boldsymbol{a} \cdot \boldsymbol{x} + \boldsymbol{b}, \qquad \boldsymbol{x} \in K. \tag{2.40}$$

This means that if the true solution of (2.38) describes a linear deformation state then it is contained in $\mathcal{W}$ and we will get an exact approximation.

2.4.1 Kinematic decomposition of $\mathcal{W}_K$

As for the finite-element method, the bilinear form $a(\cdot, \cdot)$ is divided into element contributions $a^K(\cdot, \cdot)$. To define a discrete verison of $a^K(\cdot, \cdot)$ that is exact for linear deformation states, the virtual-element space $\mathcal{W}_K$ is decomposed into three kinematically different spaces. As discussed above, the space of rigid body motions $\mathcal{R}$ contains constant deformation functions representing translation and linear deformation functions representing rotation. Hence, for a deformation state in this space the material has no strain energy. The space of constant strain modes $\mathcal{C}$ consists of linear deformation functions representing constant axial and shear strains, i.e., all deformations $\boldsymbol{u} \in \mathcal{W}_K$ that has a constant symmetric gradient $\boldsymbol{\varepsilon}(\boldsymbol{u})$. Thus, for a non-trivial deformation state in $\mathcal{C}$, the material stores potential strain energy. The last space $\mathcal{H}$ contains the higher-order polynomials and non-polynomials. The three spaces are each associated with a projection map

$$\pi_{\mathcal{R}} : \mathcal{W}_K \to \mathcal{R} \qquad \pi_{\mathcal{C}} : \mathcal{W}_K \to \mathcal{C} \qquad \pi_{\mathcal{H}} : \mathcal{W}_K \to \mathcal{H}. \tag{2.41}$$

Given this kinematic decomposition of the function space, a discrete version of the bilinear form $a^K(\cdot, \cdot)$ can be defined by its action on the different subspaces.

$$a_h^K(\boldsymbol{u}, \boldsymbol{v}) = a^K(\pi_{\mathcal{C}}\boldsymbol{u}, \pi_{\mathcal{C}}\boldsymbol{v}) + s^K(\pi_{\mathcal{H}}\boldsymbol{u}, \pi_{\mathcal{H}}\boldsymbol{v}). \tag{2.42}$$

The first term on the right-hand side in (2.42) accounts for the energy due to the constant strain modes. This can be computed exactly using only geometric information of the element. From the definition of $a(\cdot, \cdot)$ in (2.16) and the constitutive law we get

$$a^K(\pi_{\mathcal{C}}\boldsymbol{u}, \pi_{\mathcal{C}}\boldsymbol{v}) = \int_K C\boldsymbol{\varepsilon}\left(\pi_{\mathcal{C}}\boldsymbol{u}\right) : \boldsymbol{\varepsilon}\left(\pi_{\mathcal{C}}\boldsymbol{v}\right). \tag{2.43}$$

By definition of the projection map $\pi_{\mathcal{C}}$, the integrand on the right hand side is constant. Thus, using Voigt's notation we can compute this as

$$a^K(\pi_{\mathcal{C}}\boldsymbol{u}, \pi_{\mathcal{C}}\boldsymbol{v}) = |K|\boldsymbol{c}_u^T C \boldsymbol{c}_v, \tag{2.44}$$

where $|K|$ is the volume of the element and $\boldsymbol{c}_u$ and $\boldsymbol{c}_v$ are the constant strains associated with the projection of $\boldsymbol{u}$ and $\boldsymbol{v}$, respectively, into the space $\mathcal{C}$.
The difference from finite-element, when constructing the system matrix A is the second term in (2.42), $s^K(\cdot,\cdot)$. This is a prescribed symmetric positive definite bilinear form that contains the energy associated with the higher order and non-polynomial part of $\boldsymbol{u}$ and $\boldsymbol{v}$. As explained in [19] there is flexibility in the particular choice of $s^K(\cdot,\cdot)$ and a simple weighted nodal evaluation is sufficient in many cases. When constructing a system matrix, the bilinear form $s^K(\cdot,\cdot)$ will result in an addition of a symmetric positive definite matrix, i.e., the elemental system matrix is formed

$$\begin{aligned} \left(A^K\right)_{ij} =& a^K(\pi_{\mathcal{C}}\boldsymbol{\varphi}_i, \pi_{\mathcal{C}}\boldsymbol{\varphi}_j) + s^K(\pi_{\mathcal{H}}\boldsymbol{\varphi}_i, \pi_{\mathcal{H}}\boldsymbol{\varphi}_j) && (2.45)\\ =& \left(A_C^K\right)_{ij} + \left(S^K\right)_{ij}, && (2.46) \end{aligned}$$

where A_C^K and S^K are two symmetric positive definite matrices.

2.5 Matlab Reservior Simulation Toolbox

Matlab Reservoir Simulation Toolbox (MRST) [34, 17, 33] is an open source toolbox for reservoir modeling and simulation, developed at SINTEF ICT, department of Applied Mathematics. It contains a large set of routines for grid processing, numerical discretization and visualization. It is intended for easy prototyping and
The grid structure in MRST has a general storage format, giving it the ability to handle irregular grids consisting of polyhedral cells. Physical properties, such as the material stiffness tensor C, can be assigned to each grid cell.
A virtual-element method code for numerical discretization of the elasticity equation have been implemented by researchers at SINTEF ICT, as part of a geomechanics module, giving a method of modeling geomechanics on the irregular grids that are often used for reservoir simulations. For most of the numerical work presented here, this implementation of the virtual-element method and the MRST framework in general, have been used.

Chapter 3

Iterative linear solvers

Here, we turn our attention to the problem of solving the linear system that resulted from the discretizations above. We want to solve the system

$$Ax = b, \tag{3.1}$$

where A is a symmetric positive definite matrix with n rows. Let x^* denote the exact solution. There are generally two ways of attacking this problem. Either by a direct method, which results in the exact solution, or by an iterative method which gives an approximate solution. For small problems, say less than 10^4 unknowns, a direct method is undoubtedly the method of choice. For larger systems such a method can quickly become useless. Direct methods are essentially ways of performing Gaussian elimination, which are built on LU-factorization techniques. The work needed depends on how the matrix is stored. If it is stored as a general dense matrix a complete LU-factorization would require $\mathcal{O}(n^3)$ operations. If the matrix is sparse enough it is better to represent it as a sparse matrix, i.e., storing only nonzero values and the corresponding row and column indices. Taking advantage of this representation other more efficient methods have been developed. For example, matrices from regular FEM meshes can be solved with nested dissection [21] which is $\mathcal{O}(n^{3/2})$ in 2D and $\mathcal{O}(n^2)$ in 3D. These complexities are proven to be a lower bound for performing Gaussian elimination on such matrices [25].

With iterative methods we sacrifice some of the precision in order to come up with a solution faster. There are a variety of different iterative methods, some for general problems while other are optimized for specific problems. When the matrix is symmetric and positive definite, as we assume in the present work, very good alternatives to Gaussian elimination exist.

The discussion below will start with classical iterative methods, also called stationary methods. These are relatively simple methods that by themselves are not very efficient. They will however serve as important building blocks for more efficient solvers. In particular the multigrid method is built from the stationary methods. This can form a very efficient solver when combined with a Krylov subspace method, such as conjugate gradient. All of these concepts will be discussed below.

3.1 Stationary iterative methods

Classical iterative methods [15, 43] are constructed by partitioning A into different parts

$$A = M - N, \tag{3.2}$$

where the parts satisfy specific requirements. First of all M has to be nonsingular. Secondly $M^{-1}N$ has to be a contraction mapping. Using this partition we can then set up the recursion scheme

$$Mx_{k+1} = Nx_k + b, \tag{3.3}$$

noting that the solution of the original problem will also be a solution of this equation. The matrix M is called a preconditioner and is designed to approximate A and at the same time be inexpensive to invert. Choosing a starting point x_0, one can iterate towards the solution using

$$x_{k+1} = M^{-1}(N)x_k + M^{-1}b \tag{3.4}$$

until a satisfactory error threshold is reached. From this it is evident that M need not be explicitly given. Instead only the inverse can be provided. Classical iterative methods are constructed by choosing different partitions in (3.2). The most common method uses the partition

$$A = D - L - U, \tag{3.5}$$

where D is the diagonal part of A, and L and U are the negative of the strictly lower and strictly upper part of A, respectively.

3.1.1 Convergence

The equation (3.4) can be written in the form

$$x_{k+1} = Gx_k + f, \tag{3.6}$$

where $G = M^{-1}N$ is the iteration matrix and $f = M^{-1}b$ a constant vector. Let x^* be the solution of (3.1). Then x^* will solve the equation

$$x^* = Gx^* + f. \tag{3.7}$$

By subtracting (3.7) from (3.6) we get

$$x_{k+1} - x^* = G(x_k - x^*). \tag{3.8}$$

Introducing the error at a given step $e_k = x_k - x^*$ we get an expression for the error propagation,

$$e_k = G^k e_0. \tag{3.9}$$

Convergence of an iterative method can be defined as

$$\lim_{k\to\infty} \|e_k\| = 0. \tag{3.10}$$

Form some norm $\|\cdot\|$ on R^n. The Euclidean norm is used here for a simple relation to the eigen-spectrum, but because of norm equivalence, any norm on $\mathbb{R}^n$ could be used. Using (3.9) and the sub-multiplicative property of an induced matrix norm we get that

$$\|e_k\| = \|G^k e_0\| \leq \|G^k\|\|e_0\| \leq \|G\|^k \|e_0\|. \tag{3.11}$$

As the matrix norm induced by the Euclidean norm equals the spectral radius $\rho(G)$, we have

$$\|e_k\| \leq \rho(G)^k \|e_0\|. \tag{3.12}$$

This eigenvalue dependence motivates the classical convergence theorem [43] saying that (3.6) converges for any f and any x_0 if, and only if $\rho(G) < 1$. Once the iterative method is proved to converge, we want to say something about how fast it converges. From (3.12) we can also conclude that the spectral radius is an upper bound for the rate of convergence, i.e.,

$$\frac{\|e_{k+1}\|}{\|e_k\|} \leq \rho(G). \tag{3.13}$$

For each iteration the component of the error corresponding to the largest eigenvalue will become more dominant. In the limit $k \to \infty$, the error reduction will be exactly this eigenvalue. We therefore have that the asymptotic convergence rate

$$s := \lim_{k\to\infty} \frac{\|e_{k+1}\|}{\|e_k\|} = \rho(G). \tag{3.14}$$

For estimating the convergence rate from a set iteratates we can use the average convergence rate using

$$s = \left(\frac{\|e_k\|}{\|e_0\|}\right)^{\frac{1}{k}}. \tag{3.15}$$

3.1.2 Jacobi and ω-Jacobi

The simplest, Jacobi's method, uses only the diagonal part D as preconditioner, which inverts trivially. The recursion scheme for Jacobi's method will be

$$x_{n+1} = D^{-1}(L+U)x_n + D^{-1}b. \tag{3.16}$$

Hence the iteration matrix for Jacobi's method is

$$G_J = D^{-1}(L+U) = I - D^{-1}A. \tag{3.17}$$

The convergence property of this method is generally slow as some eigenvalues tend to be close to either 1 or -1. A sufficient condition for the convergence for the Jacobi method is diagonal dominance of A [30]. The discretizations for the elasticity equation presented above result in symmetric positive definite matrices, not necessarily diagonally dominant. A modification, the weighted Jacobi method or simply ω-Jacobi, changes the iteration matrix to !!!

$$G_{\omega J} = I - \omega D^{-1}A, \tag{3.18}$$

where $0 < \omega < 1$. Doing this modifies all eigenvalues to become closer to 1. For Poisson's problem the eigenmodes can be analytically found, and correspond to different frequencies [10]. The largest eigenvalues of A turns out to be those of high frequencies. These are the ones that potentially gives unstable iterations as they can result in eigenvalue of $G_{\omega J}$ below -1. By choosing $\omega < 0$ these eigenvalues can be forced into the interval $(-1, 1)$, resulting in convergence. This does not come without cost, however. The eigenvalues at the other end of the spectrum, corresponding to low frequencies, will become even closer to one for $\omega < 1$, which means that these components of the error are reduced slower. In Figure 3.1 eigenvalues for $G_{\omega J}$ are plotted for an isotropic cartesian VEM discretization of the 2D elasticity equation. As seen, the pure Jacobi method is not convergent for this particular discretization, as G_J has eigenvalues below -1. Stability is achieved using $\omega < 1$ to damp all eigenvalues so that $\rho(\omega D^{-1}A) < 2$. For this particular problem we can see that using $\omega < 0.9$ all eigenvalues of $G_{\omega J}$ are in the interval $(-1, 1)$. In Figure 3.2 the convergence histories for the same problem are shown. If ω gets too small the convergence rate will suffer as λ_{max} gets closer to 1, which can be seen in Figure 3.2. If ω-Jacobi is considered as a standalone solver one would choose ω such that $\lambda_{min} = -\lambda_{max}$ to get a spectral radius as small as possible.

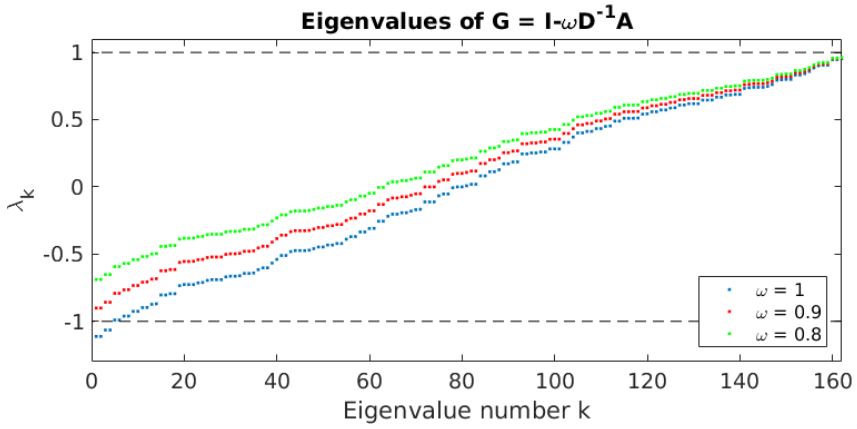


Figure 3.1: Eigenvalue distribution for the iteration matrix in the ω-Jacobi method. The dashed black lines marks the region of stability. The elasticity equation is discretized using the virtual-element method on a 10×10 grid.

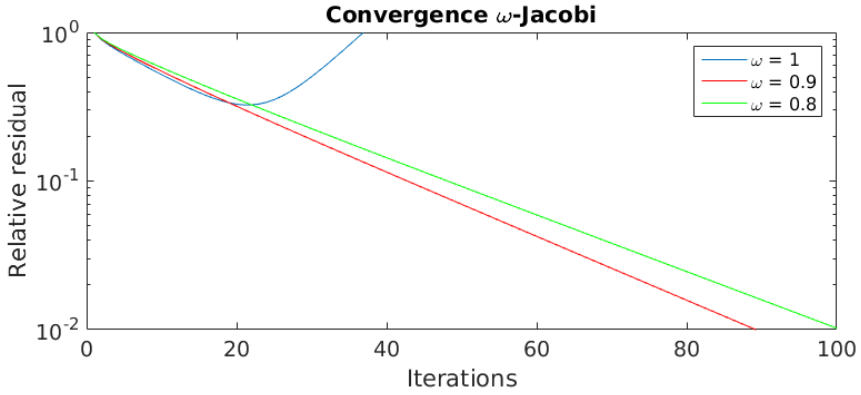


Figure 3.2: Convergence history for the ω-Jacobi method applied to the system from Figure 3.1.

3.1.3 Gauss-Seidel

The Gauss-Seidel iteration uses the lower and diagonal parts as preconditioner, giving the scheme

$$x_{n+1} = (D - L)^{-1} U x_n + (D - L)^{-1} b, \tag{3.19}$$

which gives the iteration matrix

$$G_{GS} = (D - L)^{-1} U. \tag{3.20}$$

Note that the preconditioner $M = D - L$ is triangular and inverts easily. In Jacobi's method all the vector elements are updated simultaneously using only values at the previous step. The Gauss-Seidel iteration updates one element at the time, starting with the first, taking into account the already updated elements at the current step. Similarly one gets the *backward* Gauss-Seidel iteration by letting $M = D - U$. One can also use a combination of the forward and backward iteration by sequentially taking a forward and backward step. This combination is often called the *symmetric* Gauss-Seidel (SGS) method.

3.1.4 Successive Over Relaxation

The Gauss-Seidel method is guaranteed to converge for all symmetric positive definite matrices, in contrast to Jacobi for which diagonal dominance is a sufficient condition. As we only consider symmetric positive definite matrices, this means that in stead of using a stabilizing parameter $\omega < 1$ as for the Jacobi method, we can use an *over-relaxation* parameter $\omega > 1$ to accelerate the convergence. We now form the

partition

$$\omega A = (D - \omega L) - (\omega U + (1 - \omega)D). \tag{3.21}$$

From this the *Successive Over Relaxation* (SOR) method is constructed as

$$x_{n+1} = (D - \omega L)^{-1}[\omega U + (1 - \omega)D]x_n + (D - \omega L)^{-1}\omega b, \tag{3.22}$$

which results in the iteration matrix

$$G_{SOR} = I - \left(\frac{D}{\omega} - L\right)^{-1} A. \tag{3.23}$$

Note that when $\omega = 1$ this reduces to the Gauss-Seidel iteration.
In Figure 3.3 the eigenvalue distribution of G_{SOR} is shown for different values of ω on a 2D isotropic linear elasticity problem. The pure Gauss-Seidel method, i.e., $\omega = 1$, has a very uneven distribution of eigenvalues, and a large spectral radius. By increasing ω towards 2 the eigenvalues become more uniform in terms of moduli, which means that the different eigenmodes are reduced at a more even rate. As Figure 3.3 shows $\omega = 1.8$ gives a significant reduction in spectral radius compared to $\omega = 1$. In Figure

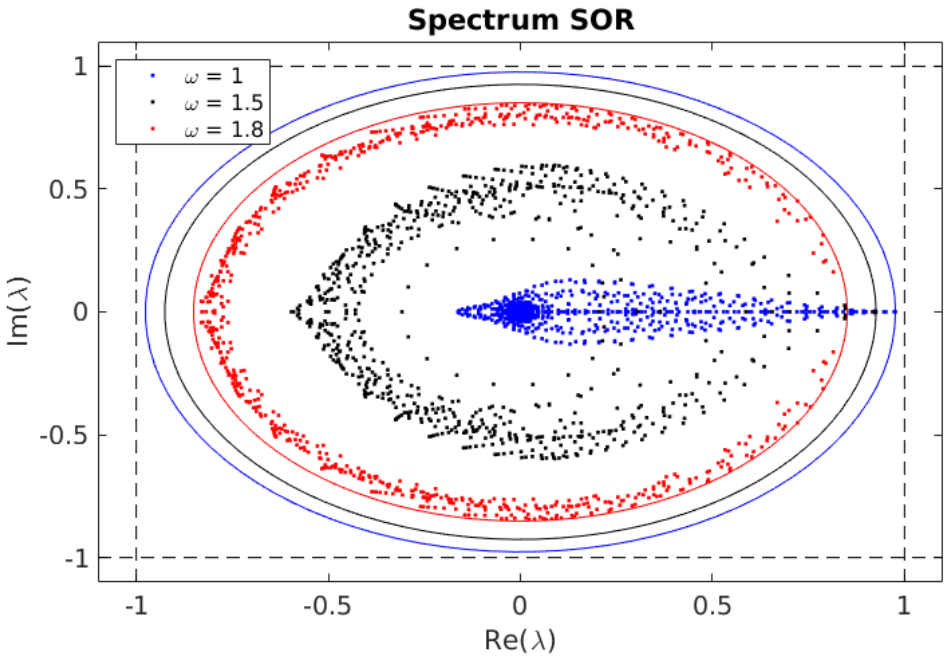


Figure 3.3: Spectrum of the iteration matrix G_{SOR} for different over-relaxation parameters ω. The solid lines marks the spectral radii. A 20×20 grid us used.

3.4 the number of iterations needed to reduce the residual norm by a factor of 10^{-4}, is shown for different values of ω. A minimum appears to be around $\omega = 1.75$.

Similarly as for SGS, the *Symmetric Successive Over Relaxation* (SSOR) method consist of sequentially taking a forward and backward step of SOR. By taking half-steps one can obtain a single recurrence scheme as in (3.4). The SSOR iteration matrix reads

$$G_{SSOR} = I - \frac{2 - \omega}{\omega}\left[\left(\frac{D}{\omega} - U\right)^{-1} D \left(\frac{D}{\omega} - L\right)^{-1}\right] A. \tag{3.24}$$

Figure 3.5 shows the convergence of the stationary iterative methods described above applied to an elasticity problem on a 60×60 grid. The value $\omega = 0.8$ for the ω-Jacobi method has been used as

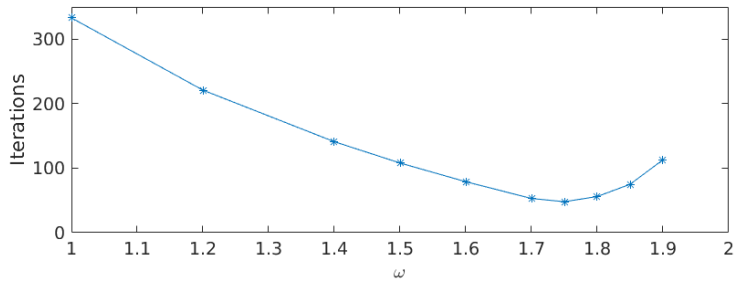


Figure 3.4: Number of iterations needed to reduce the residual norm by 10^{-4} for the SOR method with different values of ω. A 20×20 grid us used.

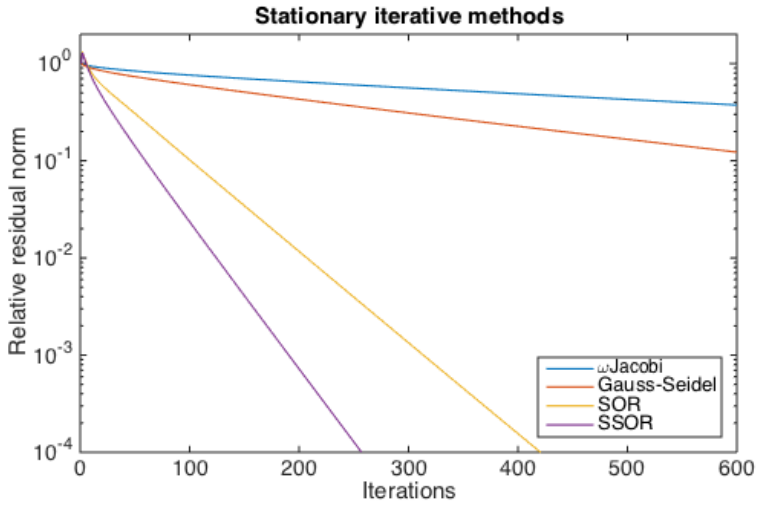


Figure 3.5: Convergence histories for stationary iterative methods applied to an isotropic 2D elasticity problem on a 60×60 grid. A value $\omega = 0.8$ is used for ω-Jacobi while the value $\omega = 1.75$ is used for SOR and SSOR.

values closer to 1 were unstable and smaller values gave slower convergence. As can be seen the average convergence rate, $s = 0.999$ for ω-Jacobi and $s = 0.997$ for Gauss-Seidel, is relatively slow. A significant improvement is seen using over-relaxation, giving $s = 0.979$ for SOR and $s = 0.966$ for SSOR.

3.2 Conjugate Gradient

For symmetric positive definite linear systems the conjugate gradient method [37, 44, 16, 32] is one of the most popular iterative solution algorithms. It is a so-called Krylov subspace method. A Krylov subspace of k-th order generated by A and b is defined as

$$\mathcal{K}_k(A, b) = \text{span}\{b, Ab, ..., A^{k-1}b\}. \tag{3.25}$$

A solution method of this class searches for a solution in this vector space. From the Cayley-Hamilton theorem [27] one can conclude that if A is an $n \times n$ invertible matrix then the n-th order Krylov subspace $\mathcal{K}_n$ equals the whole $\mathbb{R}^n$. This means that a solution found in $\mathcal{K}_n$ will be the exact solution of the original problem, hence the method can be considered a direct solver. This is indeed what Hestenes and Steifel [24] proposed when they first introduced the conjugate gradient method.

The linear equation (3.1) can be stated as the following minimization problem:

$$\min \phi(x) := \frac{1}{2}x^T Ax - b^T x, \tag{3.26}$$

As A is symmetric positive definite, $\phi(x)$ is a convex function. This means that the unique minimum of (3.26) is found where

$$\nabla\phi(x) = Ax - b = 0, \tag{3.27}$$

which shows that (3.26) and (3.1) are equivalent. For a given x_k, define the residual

$$r_k = Ax_k - b. \tag{3.28}$$

The conjugate gradient method finds x_k as the minimizer of $\phi(x)$ over the space $\{x_0 + \mathcal{K}_k(A, r_0)\}$, where x_0 is the initial iterate. At each step, a *search direction* $p_k \in \{x_0 + \mathcal{K}_k(A, r_0)\}$ is found which is A-conjugate with all the previous search directions, meaning $p_k^T Ap_i = 0$ for $i = 1, ..., k-1$. By finding the one-dimensional minimizer along this search direction, one gets, at each step the component of the final solution along the vector p_k.

The initial search direction is simply the negative of the initial residual, i.e., the negative of the gradient.
Given an iterate x_k and the search direction p_k one can find the one-dimensional minimizer, i.e., the next iterate as

$$x_{k+1} = x_k + \alpha_k p_k, \tag{3.29}$$

where

$$\alpha_k = \frac{r_k^T r_k}{p_k^T Ap_k}. \tag{3.30}$$

The problem is now to find the next search direction. An important observation is that the new residual r_{k+1} must be linearly independent with all the previous search directions. This can then be used, together with the previous search direction, to construct the next direction as

$$p_{k+1} = -r_{k+1} + \beta_{k+1}p_k, \tag{3.31}$$

where

$$\beta_{k+1} = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}. \tag{3.32}$$

Derivation of the scalars α_k and β_k can be found in [37]. The resulting procedure is shown in Algorithm 1.

Algorithm 1 Conjugate gradient

Given x_0

Set $r_0 = Ax_0 - b$, $p_0 = -r_0$, $k = 0$

while $r_k \neq 0$ **do**

  $\alpha_k = \frac{r_k^T r_k}{p_k^T A p_k}$

  $x_{k+1} = x_k + \alpha_k p_k$

  $r_{k+1} = r_k + \alpha A p_k$

  $\beta_{k+1} = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$

  $p_{k+1} = -r_{k+1} + \beta_{k+1} p_k$

  $k = k + 1$

end while

Even though the conjugate gradient method finds the exact solution in at most n iterations it is more commonly regarded as an iterative method, i.e., one stops iterating when it has reached a desired error threshold. For this we need some way of measuring the error at each iterate. Since we do not know the exact solution we can not measure the exact error. Instead we can use the residual (3.28), which, at the exact solution is zero. A conventional choice is therefore to stop iterating once the 2-norm $\|r\| = (r^T r)^{1/2}$ gets below the desired threshold.

For this approach it is interresting to examine the convergence properties in order to estimate its efficiency as an approximation method.

3.2.1 Convergence of conjugate gradient

The goal of this section is to establish a relation between the initial error and the error at the k-th iterate, and in particular relate this to the condition number $\kappa := \lambda_{max}/\lambda_{min}$ of A.
At the iteration k we have minimized the components of the error along the directions $p_1, p_2, ..., p_k$. This means that

$$\|x_k - x^*\|_A = \inf_{v \in \mathcal{S}_k} \|x^* - (x_0 - v)\|_A, \tag{3.33}$$

where

$$\mathcal{S}_k = \mathcal{K}_k(A, r_0), \tag{3.34}$$

is the k-th order Krylov subspace generated by A and r_0. So iterating up to $x_k = x_0 - v$ we have found the optimal coefficients a_i of the linear combination

$$v = \sum_{i=1}^{k} a_i A^{i-1} r_0 = p_{k-1}(A) r_0. \tag{3.35}$$

The last term is just a change of notation introducing the matrix polynomial $p_{k-1}(A)$ of degree $k-1$. Using the residual equation $Ae = r$ we can represent this in terms of the initial error as

$$v = A p_{k-1}(A) e_0 = A p_{k-1}(A)(x_0 - x^*). \tag{3.36}$$

With this representation we can rewrite (3.33) as

$$\|x_k - x^*\|_A = \inf_{p \in \mathcal{P}_{k-1}} \|(I - Ap(A))(x^* - x_0)\|_A, \tag{3.37}$$

where $\mathcal{P}_{k-1}$ denotes the space of all polynomial of degree $k-1$. Because A is symmetric positive definite we can bound the norm on the right hand side using the 2-norm as

$$\|x_k - x^*\|_A \leq \inf_{p\in\mathcal{P}_{k-1}} \|(I - Ap(A))\|_2 \|(x^* - x_0)\|_A. \tag{3.38}$$

To ease notation let us define the constant

$$C_k = \inf_{p\in\mathcal{P}_{k-1}} \|I - Ap(A)\|_2, \tag{3.39}$$

so that we can write

$$\|x_k - x^*\|_A \leq C_k \|(x^* - x_0)\|_A. \tag{3.40}$$

As the 2-norm equals the spectral radius, this constant can be written using the eigenvalues of A. If λ is an eigenvalue of A then $p(\lambda)$ is an eigenvalue of $p(A)$, where $p(\cdot)$ now denotes both a matrix polynomial and the scalar polynomial with the same coefficients. Hence

$$C_k = \inf_{p\in\mathcal{P}_{k-1}} \max_{1\leq j\leq n} |1 - \lambda_j p(\lambda_j)|. \tag{3.41}$$

The conjugate gradient method automatically finds the best polynomial for the eigenvalue that maximizes $|1 - \lambda_j p(\lambda_j)|$. The problem when estimating the error reduction is that we generally do not know the eigenvalue distribution. Instead of finding and evaluating the expression at each eigenvalue we can consider the interval of which the eigenvalues are contained in, and thus convert it to a continuous problem. Let λ_1 be the smallest eigenvalue and λ_n be the largest. As the interval $[\lambda_1, \lambda_n]$ contains all the eigenvalues we get the inequality

$$C_k \leq \inf_{p\in\mathcal{P}_{k-1}} \max_{\lambda_1\leq x\leq\lambda_n} |1 - xp(x)|. \tag{3.42}$$

Let us simplify this expression by introducing a new, k-th order polynomial $q(x) := 1 - xp(x)$. Notice that $q(0) = 1$ no matter what the polynomial p is. With this we can write

$$C_k \leq \inf_{\substack{q\in\mathcal{P}_k \\ q(0)=1}} \max_{\lambda_1\leq x\leq\lambda_n} |q(x)|. \tag{3.43}$$

This continuous expression can be calculated explicitly using Chebyshev polynomials. The k-th Chebyshev polynomial is defined as $T_k(y) := \cos(k \arccos y)$. It can be shown, see [2], that the polynomial

$$Q_k(x) = \frac{T_k\left(\frac{\lambda_n+\lambda_1-2x}{\lambda_n-\lambda_1}\right)}{T_k\left(\frac{\lambda_n+\lambda_1}{\lambda_n-\lambda_1}\right)} \tag{3.44}$$

is the minimizer of (3.43). As $T_k(y)$ oscillates between -1 and 1 we can bound the numerator by 1 and estimate $|Q_k(x)|$ using only the denominator. Hence

$$C_k \leq \left[T_k\left(\frac{\kappa+1}{\kappa-1}\right)\right]^{-1}, \tag{3.45}$$

introducing the condition number $\kappa = \lambda_n/\lambda_1$. By the definition of T_k this can be rewritten

$$C_k \leq \frac{2}{\left(\frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1}\right)^k + \left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k}. \tag{3.46}$$

As the condition number is always greater than 1, the second term in the denominator tends towards zero as $k \to \infty$. We can thus suppress this term and arrive at the final estimation of the error reduction for the conjugate gradient method,

$$\|x_k - x^*\|_A \leq 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k \|x_0 - x^*\|_A. \tag{3.47}$$

This describes linear convergence with rate

$$r = \frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}. \tag{3.48}$$

An important observation here is that for large condition numbers the value of r increases towards 1, which slows down the convergence. In such a case we say that the matrix is *ill-conditioned*.
Although this only describes the "worst-case" performance of the conjugate gradient method, it often gives a prediction of how the method actually performs.

3.2.2 Preconditioned Conjugate Gradient

To improve the convergence rate we can look at a preconditioned system. Assume that we have a symmetric positive definite matrix M that is thought to approximate A well. A preconditioned version of (3.1) would then be

$$M^{-1}Ax = M^{-1}b. \tag{3.49}$$

A solution of this equation will also be a fixed point of the classical iteration scheme (3.3), which explains why the term preconditioner is used in both settings. The matrix could for example be the diagonal of A, in which case we would, by applying fixed point iterations, recover the Jacobi method.
Even though both M and A are symmetric the system matrix of (3.49) is generally not symmetric, which is required for the conjugate gradient method. We can instead use another preconditioning approach that preserves symmetry. When M is symmetric positive definite we can split it as $M = M^{\frac{1}{2}}M^{\frac{1}{2}}$ where $M^{\frac{1}{2}}$ (and $M^{-\frac{1}{2}}$) is spd as well.
Consider now instead the system

$$M^{-\frac{1}{2}}AM^{-\frac{1}{2}}\hat{x} = \hat{b}, \tag{3.50}$$

where $\hat{x} = M^{\frac{1}{2}}x$ and $\hat{b} = M^{-\frac{1}{2}}b$. Now the system matrix is symmetric positive definite. The reason for using the square root of M and not just M itself is to simplify the expressions resulting when we apply the conjugate gradient method.
Let us express the iteration procedure for this system using the original vectors x and b, by inserting into the expressions above. The direction vectors will scale in the same way as x, i.e., $\hat{p} = M^{\frac{1}{2}}p$ and the residual will scale the same way as the right hand side, i.e., $\hat{r} = M^{-\frac{1}{2}}r$.
For the iteration step (3.29) each term scales the same, which leaves it unchanged. As p and r scale differently, the update of the search direction (3.31) will change to

$$p_{k+1} = -M^{-1}r_{k+1} + \beta_{k+1}p_k. \tag{3.51}$$

The expressions for the scalars α and β will now be

$$\alpha_k = \frac{\hat{r}_k^T\hat{r}_k}{\hat{p}_k^T M^{-\frac{1}{2}}AM^{-\frac{1}{2}}\hat{p}_k} = \frac{r_k^T M^{-1}r_k}{p_k^T Ap_k} \tag{3.52}$$

and

$$\beta_{k+1} = \frac{\hat{r}_{k+1}^T\hat{r}_{k+1}}{\hat{r}_k^T\hat{r}_k} = \frac{r_{k+1}^T M^{-1}r_{k+1}}{r_k^T M^{-1}r_k}. \tag{3.53}$$

One can notice that the only difference in this preconditioned version compared to the original is that r has changed to $M^{-1}r$ at three places. So we introduce the preconditioned residual

$$z = M^{-1}r. \tag{3.54}$$

The resulting procedure is shown in Algorithm 2.
Observe that the matrix M is not used in the procedure, only its inverse, which means we do not need to know M explicitly. In fact we do not need to know M^{-1} either, we just have to solve the equation $Mz = r$, which means that we can provide the preconditioner as a black-box solver instead of an explicit matrix.

Since we now solve the preconditioned system (3.50) we get a different value for the convergence rate (3.48). It now depends on the condition number of the preconditioned system matrix $M^{-\frac{1}{2}}AM^{-\frac{1}{2}}$. Notice that the symmetric preconditioning (3.50) results in the same upper bound for the convergence rate as the left preconditioning (3.49). If (λ, v) is an eigenpair of $M^{-1}A$ then $(\lambda, M^{\frac{1}{2}}v)$ is an eigenpair of $M^{-\frac{1}{2}}AM^{-\frac{1}{2}}$, which means that the two systems have identical eigenvalues, and the condition number wil be the same in both cases. With a good preconditioner all the eigenvalues of the preconditioned system should be close to 1, which would give a condition number $\kappa = \lambda_n/\lambda_1$ not much greater than 1. With this we can see from (3.47) that the convergence rate improves drastically.

Algorithm 2 Preconditioned Conjugate Gradient

Given x_0 and preconditioner M

Set $r_0 = Ax_0 - b$

Solve $Mz_0 = r_0$ for z_0

Set $p_0 = -z_0$, $k = 0$

1 **while** $r_k \neq 0$ **do**

2 $\quad \alpha_k = \frac{r_k^T z_k}{p_k^T A p_k}$

3 $\quad x_{k+1} = x_k + \alpha_k p_k$

4 $\quad r_{k+1} = r_k + \alpha A p_k$

5 $\quad$ Solve $Mz_{k+1} = r_{k+1}$ for z_{k+1}

6 $\quad \beta_{k+1} = \frac{r_{k+1}^T z_{k+1}}{r_k^T z_k}$

7 $\quad p_{k+1} = -z_{k+1} + \beta_{k+1} p_k$

8 $\quad k = k + 1$

9 **end while**

3.3 Matrix based preconditioners

As evident above the objective when applying a preconditioner to the conjugate gradient method is to reduce the condition number κ. The "perfect" preconditioner would be A itself, in which case the condition number would be one, and the error would be reduced to zero in one iteration. The use of this preconditioner is of course practically pointless because it involves the inversion A. Instead we find a matrix M that approximates A, so that the matrix $M^{-1}A$ has a condition number as close to one as possible.

3.3.1 Jacobi, SOR preconditioners

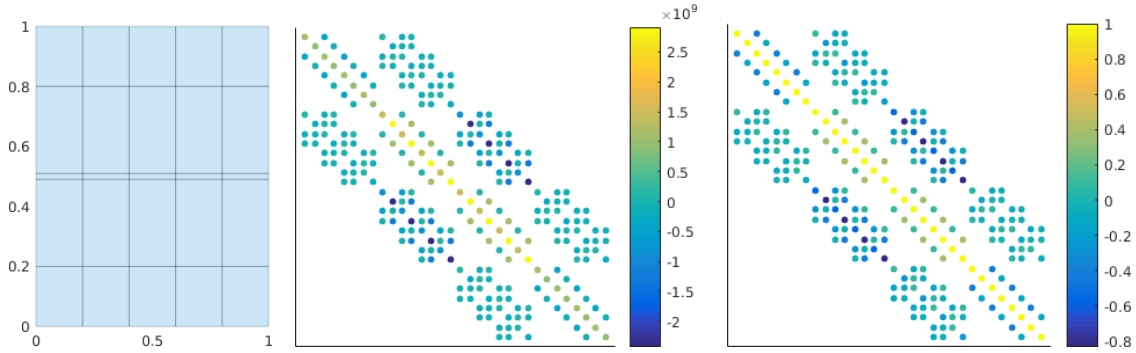
A simple way to make preconditioners are to use the same idea as for the stationary iterative methods above. Let us assume A is symmetric positive definite so we can partition it into a diagonal and a lower and upper triangular part,

$$A = D - L - L^T. \tag{3.55}$$

We will refer to preconditioners based on this matrix splitting as *splitting preconditioners*. It is worth noticing that from Sylvester's criterion [28] for symmetric positive definite matrices we can conclude that the diagonal matrix D has only strictly positive values.
Let us first consider a Jacobi preconditioner,

$$M = D. \tag{3.56}$$

The effect of this preconditioner is a scaling of each row by the inverse of its diagonal element so that the diagonal of $M^{-1}A$ becomes the identity matrix. This will have a positive effect if the system is irregular in terms of grid or PDE-coefficients. In that case some unknowns have a larger "self-energy", i.e., $a(\boldsymbol{\varphi}_i, \boldsymbol{\varphi}_i)$, than others. The diagonal scaling will diminish this difference and consequently produce a more even distribution of eigenvalues. An illustrative example is the irregular grid shown in Figure 3.6a. The sparsity pattern of the virtual element system matrix on this system is depicted in Figure 3.6b where the color shows the magnitude of the entries. The diagonal entries are the self-energy associated with each degree of freedom. The largest diagonal elements corresponds to nodes adjacent to the smaller grid cells, because a unit value in these unknowns, results in a steeper gradient, i.e., larger strain. In Figure 3.6c the diagonal scaling $D^{-1}A$ have been performed, producing a unit diagonal.



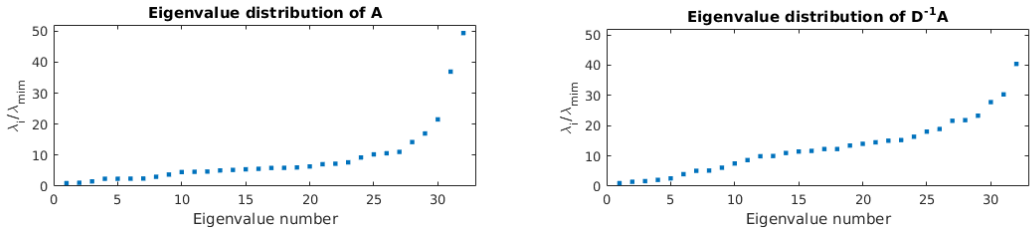
(a) Irregular grid with a large variation in cell volumes

(b) Sparsity pattern and magnitudes for A

(c) Sparsity pattern and magnitudes for $D^{-1}A$

Figure 3.6: Example of diagonal scaling on an irregular grid. Notice that all the diagonal elements of $D^{-1}A$ are identical.

In Figure 3.7 the eigenvalue distributions for A and $D^{-1}A$ are shown. As can be seen the diagonal scaling evens out the distribution and reduces the span. The condition number is reduced from 49 to 42.
The diagonal scaling is essentially equivalent to normalizing each basis function so that the self-energy $a(\boldsymbol{\varphi}_i, \boldsymbol{\varphi}_i)$ is equal for all i. This compensates for irregularities in terms of varying cell-volumes or PDE-coefficients. The scaling does not however entirely eliminates the undesired effects of an irregular grid, as off-diagonal elements will still have significant variations.
The diagonal scaling D^{-1} produces in general a non-symmetric matrix. As mentioned above one can instead use the diagonal as a split preconditioner $D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ that has the equivalent effect on eigenvalues but preserves the symmetry.
For completely regular problems the diagonal, at least away from boundaries is a multiple of the identity matrix. The use of a Jacobi preconditioner will then have no effect as it scales all matrix elements, hence



(a) **(b)**

Figure 3.7: Eigenvalues for **(a)** A and **(b)** $D^{-1}A$, scaled by $1/\lambda_{min}$. The largest value gives the condition number.

all eigenvalues equally, leaving the condition number unchanged.

The formulation of the preconditioned conjugate gradient above builds on the assumption that M is symmetric positive definite. It is therefore not justified to use for example the Gauss-Seidel method $M = D - L$ as a preconditioner, which is triangular. Instead the symmetric version, SGS, can be used, where a forward Gauss-Seidel sweep is followed by a backward sweep, which gives the resulting preconditioner

$$M = (D - L)D^{-1}(D - L^T). \tag{3.57}$$

This is clearly symmetric, and because D is strictly positive, M is also positive definite. Therefore it can be used in the preconditioned conjugate gradient method.
Adding the over-relaxation technique introduced in Section 3.1.4, we can modify the symmetric Gauss-Seidel preconditioner and get the SSOR preconditioner,

$$M = \frac{\omega}{2-\omega}\left(\frac{D}{\omega} - L\right)D^{-1}\left(\frac{D}{\omega} - L^T\right). \tag{3.58}$$

We will in the following make use of the notation SSOR(ω) for this preconditioner.

3.3.2 Incomplete factorization

One can notice that the splitting preconditioners (3.57) and (3.58) consist of a lower and an upper triangular factor, $M = L_M U_M$. In our symmetric case we can also readily extract the triangular factors that are transpose of each other, i.e., the cholesky decomposition $M = L_M L_M^T$. For SGS this factor reads

$$L_M = (D - L)D^{-\frac{1}{2}}. \tag{3.59}$$

We can then compute the error matrix for the preconditioner

$$A - M = A - L_M L_M^T = -LD^{-1}L^T. \tag{3.60}$$

This motivates the search for other factors than L_M that try to minimize the error matrix, which gives rise to a general family of preconditioners termed incomplete factorization. For general non-singular matrices this would be based on the LU-decomposition of A, which leads to the class of ILU preconditioners. For symmetric positive definite matrices, and the use of conjugate gradient, we instead consider the symmetric Cholesky factorization of A.

The Cholesky factorization of a real symmetric positive definite matrix A consists of finding the unique lower triangular matrix K such that

$$A = KK^T \tag{3.61}$$

To understand how the factorization process work it is usful to visualize the process on the matrix graph. The graph of a matrix is a collection of vertices and edges, where each vertex correspond to an unknown in the matrix equation and the edges represent couplings between unknowns, see Figure 3.8.

$$A = \begin{pmatrix} x & x & & x & x & \\ x & x & x & x & & \\ & x & x & x & x & x \\ x & x & x & x & & \\ x & & x & & x & x \\ & & x & & x & x \end{pmatrix}$$

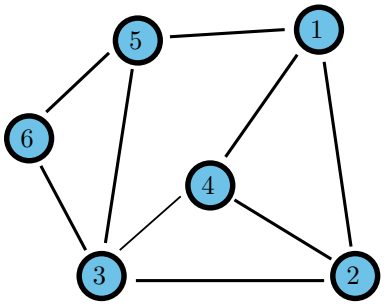


Figure 3.8: Undirected matrix graph showing the nonzero elements of a symmetric matrix. The edges corresponding to the diagonal entries are not drawn.

The process of finding K is a modified version of Gaussian elimination. In the factorization process each unknown is sequentially removed from all equations by subtracting the outer product of its row and column. The first step goes as

$$A = \begin{pmatrix} a & b^T \\ b & C \end{pmatrix} \qquad \longrightarrow \qquad \begin{pmatrix} 1 & 0 \\ 0 & C - \frac{bb^T}{a}, \end{pmatrix} \tag{3.62}$$

Giving the first column of the factorization

$$K_1 = \begin{pmatrix} \sqrt{a} & 0 \\ \frac{b}{\sqrt{a}} & I \end{pmatrix}. \tag{3.63}$$

Subsequent steps performs the same procedure on the Schur complement $C - \frac{bb^T}{a}$. The subtraction of the outer product bb^T will create fill-in in the schur complement, giving a denser matrix at each step. Figure 3.9 shows how the fill-in is added at each step following the process on the matrix graph. New couplings are created between unknowns that are coupled to the removed unknown.

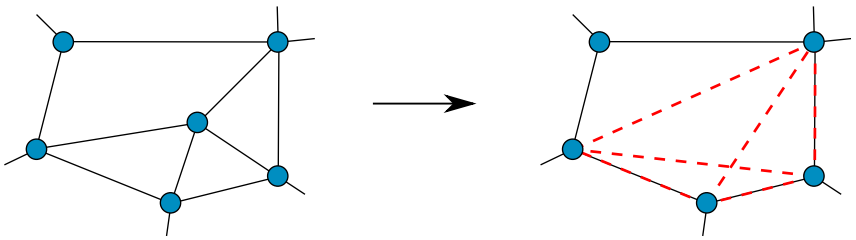


Figure 3.9: One step of a complete Cholesky factorization, removing the unknown in the middle of the graph. The red, dashed edges are the ones corresponding to the outer product bb^T.

The complete Cholesky factorization is the main ingredient in standard direct solvers for symmetric positive definite matrices, such as MATLAB's *mldivide*. For preconditioning purposes it is imperative to drastically simplify this process in terms of computational cost and memory requirements. This is usually done by restricting the fill-in created by the outer product, according to some predefined pattern. The most common approach is to restrict K to have the same non-zero structure as A. The outer product subtraction will then create no extra fill-in, only modify already existing couplings. The matrix graph will

thus get no additional edges, as in Figure 3.10. The resulting preconditioner is termed IC(0). Subsequent steps will therefore be much cheaper to perform as the matrix density does not increase.

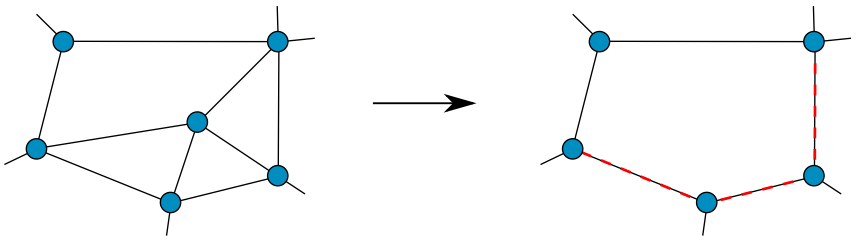


Figure 3.10: One step of the incomplete Cholesky factorization leading to the IC(0) preconditioner. Only already existing couplings are modified, all others are ignored.

Another approach to reduce the fill-in from the complete Cholesky factorization is to drop all entries that are below some threshold. With this approach one can find a suitable trade-off between the approximation quality $M \approx A$ and the cost of finding and applying the preconditioner. This threshold-based incomplete Cholesky preconditioner is often termed ICT(δ), where δ denotes the dropping threshold.

The above incomplete factorizations are not guaranteed to exist for all symmetric positive definite matrices. When subtracting the outer product bb^T it is possible to produce a zero on the diagonal of the Schur complement. If this zero is encountered as a pivot in a later step the resulting Schur complement becomes infinite and the factorization breaks down. If such a problem arise one can instead perform factorization on a diagonally shifted matrix $A_\alpha = A + \alpha I$. The value of α can be chosen so that A_α becomes a diagonally dominant matrix, for which a factorization is guaranteed to exist. The resulting factorization is generally a less accurate approximation of A, but can be a useful preconditioner nonetheless.
Another well-known family of incomplete factorization preconditioners are the modified incomplete Cholesky (MIC) preconditioners. These methods try to compensate for the dropped fill-in at each step by modifying the diagonal elements. A popular strategy is to subtract all the dropped elements from the diagonal. This results in an incomplete factorization $A \approx KK^T$ that has the same row sum as A. This means that the factorization preserves the action on the constant vector, $c = (1, 1, ..., 1)^T$, i.e.,

$$Ac = KK^T c. \tag{3.64}$$

For PDE-problems where the constant vector is assumed to be a significant component of the solution this modification can be useful as applying the preconditioner will effectively remove constant errors.

Chapter 4

Multigrid

Multigrid methods have proven to be a very efficient technique for solving large linear systems. They are based on the two concepts of *smoothing* and *coarse grid correction*, which are designed to compliment each other. The interplay between these two processes is effective in a sense that errors in different parts of the spectrum are eliminated in contrast to using only stationary iterative methods that are most effective on small eigenvalue components. The resulting method is called optimal because it requires only $\mathcal{O}(n)$ operations and its convergence rate can theoretically be independent of the problem size. The computations can be effectively distributed across parallel computers, which means that multigrid methods in theory are able to solve increasingly larger problems in essentially constant time.
Since the multigrid idea was introduced in the 60's there have been developed many different versions and improvements. The two main categories are geometric multigrid, which was how the multigrid theory originated, and algebraic multigrid [41], introduced in the 80's. In the geometric methods the different grids are generated as coarser discretizations of the domain and solutions on different grids are combined using appropriate transfer operators. This requires in general knowledge of the underlying PDE. The algebraic methods on the other hand only use information available in the system matrix to construct a suitable hierarchy of (virtual) grids and transfer operators.

A description of the underlying multigrid concepts will be given in the following, which will follow a geometric approach. Algebraic methods will be discussed in detail later.

We consider the problem (3.1) which on the original, or finest, grid Ω^h will be written

$$A^h u^h = b^h. \tag{4.1}$$

The grids are often hierarchically organized as $\Omega^h \supset \Omega^{2h} \supset ... \supset \Omega^H$, and as you move down the hierarchy smaller and smaller systems have to be solved, or smoothed.

4.1 Multigrid concepts

4.1.1 Smoothing

A multigrid method always starts with smoothing, or relaxation, of the equation. This usually means performing a few iterations of a stationary iterative method, often Jacobi or Gauss-Seidel. Such methods have the so-called smoothing property. The error can be represented as a linear combination of the eigenvectors of the iteration matrix. The smoothing terminology refers to more efficient reduction of oscillatory than smooth components of the error. The eigenvectors of stationary iteration matrices for elliptic problems correspond to different spatial frequencies [12]. For Jacobi and Gauss-Seidel the largest

eigenvalues of G, i.e., the ones closest to 1, are those corresponding to low frequencies. This means that after some iterations the smooth error components become dominant. Stationary methods, by themselves, therefore suffer from a stagnation after a few iterations. This smoothing effect is shown in Figure 4.1 for Gauss-Seidel applied to Laplace's equation. Even though SOR is more efficient than Gauss-Seidel as a stand-alone solver it does not have the same smoothing property as Gauss-Seidel. This is related to the redistribution of eigenvalues for $\omega > 1$, which is seen in Figure 3.3.

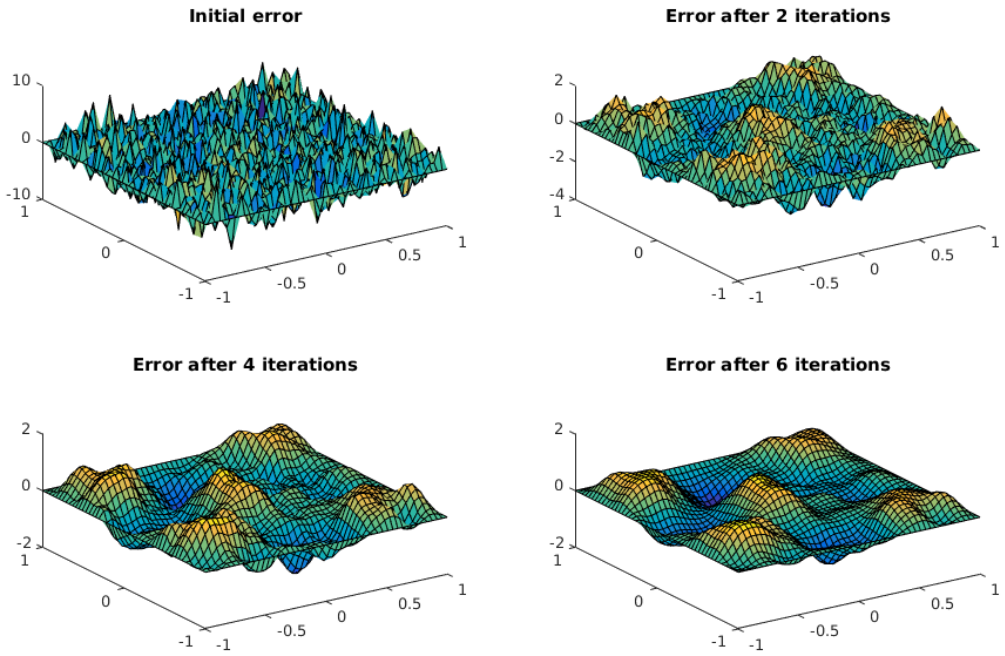


Figure 4.1: Gauss-Seidel smoothing on a normally distributed initial error for Laplace's equation.

4.1.2 Coarse grid correction

The essential ingredient of the multigrid idea is the coarse grid correction. As the name suggests this technique looks at the problem on a coarser grid, and uses the solution there to correct for errors on the original, or fine grid. The reasons for why this is a useful technique are twofold. First of all, by solving the problem on a coarser grid, i.e., a smaller system, one can take a relatively large step towards the solution on the original grid, at a low cost. Secondly, it can be designed to complement the smoothing property of stationary iterative methods. In fact, a coarse grid correction will only correct lower frequencies. To explain the latter one can consider the line segment $\Omega = [0, 1]$ discretized by n_f points giving the discrete space Ω^h. Then any vector in Ω^h can be represented by frequencies in the range 0 to $(n_f - 1)/2$. If we next make a coarser grid by discarding every other point, any vector on this grid can be represented by frequencies in the range 0 to $(n_f - 1)/4$, which means that we loose the upper half of the frequency specter on this coarser grid. By solving the problem here we would eliminate all the lower half frequency components of the error, which indeed are the ones that a stationary method would have most problems reducing.

In Figure 4.2, a 1D Poisson problem on a line segment discretized with finite difference is considered. It shows the evolution of the frequency distribution of the error at different stages of a solution process. Figure 4.2a is the frequency distribution of a normally distributed random initial error. For this, the frequency components are relatively uniformly distributed. As expected, a few iterations of Gauss-Seidel

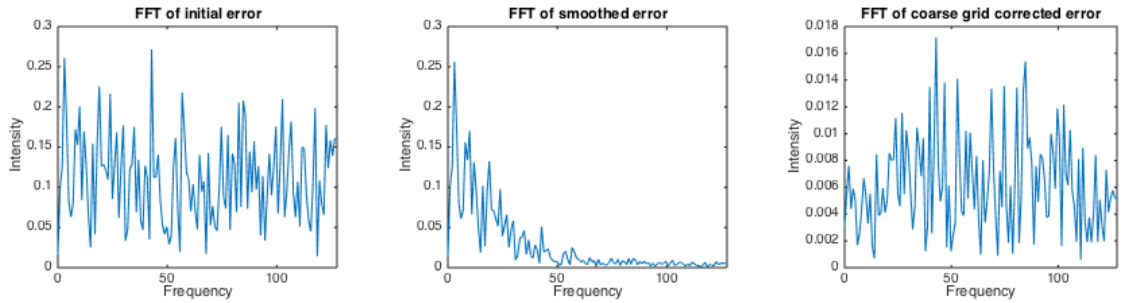


Figure 4.2: Frequency distribution of a) the initial error, b) the error after a few smoothing iterations of Gauss-Seidel and c) after correcting the error from b) on a coarser grid.

would smooth this error, in effect reducing the high frequency components more than the lower, which can be seen in Figure 4.2b. In Figure 4.2c a coarse grid correction is applied, where the problem is solved exact on the coarser grid. This reduces the low frequencies, shifting the distribution towards the right. Note that the y-scale has changed in Figure 4.2c. As can be seen, the low frequency componennts are not entirely removed, even though an exact coarse grid solution is found. This has to do with transferring information between the original and the coarse grid problem, and the fact that the coarse grid problem is only an approximation of the fine grid problem. The transferring between the grids is an important aspect of the multigrid method, and different techniques for this are essentially what separates different variants of the multigrid method. This will be discussed below, but for now assume we have a *restriction operator* $R : \Omega^h \to \Omega^H$ mapping from the basis of the fine grid to the basis of the coarse grid, and a *prolongation operator* $P : \Omega^H \to \Omega^h$ mapping back to the fine grid. Both R and P are of full rank.

Assume we have performed a number of smoothing iterations on the fine grid, and obtained u^h. Then we have an expression for the residual

$$r^h = A^h u^h - b^h \tag{4.2}$$

and the error

$$e^h = u^h - \bar{u}^h. \tag{4.3}$$

As we in general do not have the exact solution $\bar{u}^h$, the error is considered unknown. Combining (4.2) and (4.3) we get the residual equation

$$A^h e^h = r^h. \tag{4.4}$$

The coarse grid correction should result in an approximation $\tilde{e}^h$ of the fine grid error, which can be used to correct the current iterate as

$$u^h \leftarrow u^h - \tilde{e}^h. \tag{4.5}$$

This means that we want to find an approximate solution of (4.4). At this point we can turn to the variational principle, similar to when we approximated functions en $H^1(\Omega)$ by functions in the finite space V for the finite-element method. Here we want to approximate a function on Ω^h by a function in Ω^H. According to the variational principle we can state (4.4) as

$$\min_{e^h \in \Omega^h} \langle e^h, A^h e^h \rangle - \langle r^h, e^h \rangle, \tag{4.6}$$

where $\langle \cdot, \cdot \rangle$ is the Eucledian inner product. Restricting ourselves to Ω^H means that we are looking for the best approximation in range(P), i.e., we want to find the minimizer of

$$\min_{e^H \in \Omega^H} \langle Pe^H, A^h Pe^H \rangle - \langle r^h, Pe^H \rangle. \tag{4.7}$$

Since P is of full rank and A^h is symmetric positive definite we can write this problem as

$$P^T A^h P e^H = P^T r^h, \tag{4.8}$$

which gives us the linear system to be solved on the coarse grid. In most cases the restriction and prolongation operators are related as

$$R = cP^T, \tag{4.9}$$

where $c \in \mathbb{R}$ is often just equal 1. We can then write the more common formulation of the coarse grid problem

$$RA^h P e^H = Rr^h, \tag{4.10}$$

which means we define the coarse grid operator

$$A^H := RA^h P, \tag{4.11}$$

known as the Galerkin operator in the multigrid literature. It is straight-forward to show that the smoother on the coarser grid can be represented in the same way, i.e.,

$$M^H = RM^h P. \tag{4.12}$$

We can now form the two-level method by combining smoothing and coarse grid correction. The resulting method is described in Algorithm 3.

Algorithm 3 Two-level method

Given:

A - fine grid matrix
b - right hand side
M - fine grid smoother
R - restriction operator
P - prolongation operator

Procedure:

1. Apply smoother M to obtain u^h
2. Compute residual $r^h = Au^h - b$
3. Restrict residual $r^H = Rr^h$
4. Solve $A^H e^H = r^H$ with $A^H = RAP$
5. Prolongate error $\tilde{e}^h = Pe^H$
6. Correct approximation $u^h \leftarrow u^h - \tilde{e}^h$
7. Apply smoother to obtain final u^h

4.1.3 Intergrid operators

In order to perform a coarse grid correction we need to formally define the intergrid operators P and R. This task is closely related to the construction of the coarse grid, as these operators are mappings between bases of the two grids. The usual approach is therefore to start by constructing a coarse grid that is a good approximation of the fine grid while having the desired ability to represent smooth vectors well. There are different ways of attacking this coarsening issue, much depending on the type of fine grid used as well as the properties of the system to be solved. This is essentially where the separation of the

geometric and the algebraic multigrid method lies. The algebraic method will be discussed in detail later. To introduce the concepts a simple geometric approach will be considered.

Let Ω^h be a nodal discretization of a line segment by n_h nodes, shown at the top of Figure 4.3. A straight-forward construction of a coarse grid would be to discretize the same segment with twice the stepsize, which we know from the discussion of Figure 4.2 would represent smooth components well. This coarse grid is shown at the bottom of Figure 4.3. Let us first consider the prolongation. As seen in Figure 4.3 each coarse grid node overlap with a fine grid node. It is therefore natural to let values at coarse grid nodes map unchanged to their respective fine grid node. For the fine grid nodes not in the coarse grid one can use linear interpolation, as shown in Figure 4.3.

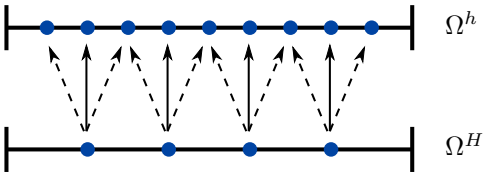


Figure 4.3: Prolongation by linear interpolation. Solid arrows indicates weights of 1 while dashed arrows indicates weights of 1/2.

The resulting prolongator will be the matrix

$$P = \begin{pmatrix} \frac{1}{2} & & & & & \\ 1 & & & & & \\ \frac{1}{2} & \frac{1}{2} & & & & \\ & 1 & & & & \\ & \frac{1}{2} & \frac{1}{2} & & & \\ & & 1 & & & \\ & & \frac{1}{2} & \ddots & \frac{1}{2} & \\ & & & & 1 & \\ & & & & \frac{1}{2} & \end{pmatrix}, \tag{4.13}$$

which is of size $n_h \times n_H$.
For defining the restriction operator operator R the most simple approach would be by injection, i.e., to copy the values of the fine grid nodes to their overlapping coarse grid nodes and ignore the other fine grid nodes. Another possibility is by full-weighting, where the non-coarse nodes also contribute to the value at their surrounding coarse nodes. This results in a smoothing effect on the restriction which is desirable. Using this we also get the nice relation between the restriction and the prolongation matrix,

$$R = \frac{1}{2}P^T = \begin{pmatrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & & & & & & \\ & & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & & & & \\ & & & & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & & \\ & & & & & & \ddots & & \\ & & & & & & & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \end{pmatrix}. \tag{4.14}$$

The above examples of intergrid operators and coarse grid constructions assume the problem at hand to be as simple as possible, i.e., an isotropic PDE with homogeneous coefficients, discretized with simple finite differences or similar on equidistant grid. On relevant problems that are more complex the resulting method might perform poorly. Efforts to modify operators to improve the performance on such cases have been done but it requires insight to the continuous problem and the discretization method, which often results in a very problem specific method. This largely motivates the algebraic multigrid method, discussed below, which is better suited for a wide range of problems.

4.1.4 Multigrid cycle

While there are different versions of the multigrid algorithm, the two base ingredients for all are smoothing and coarse grid correction. The simplest and most widely used version is the V-cycle. This is simply the recursive extension of the two-level method in Algorithm 3, i.e., instead of solving the coarse problem exactly one performs a few smoothing steps and proceeds to an even coarser grid. The recursion stops when reaching a level small enough to be efficiently solved by a direct method. The size of this coarsest level depends on the cost of further coarsening compared to the use of direct solvers. In [39] the coarsening stops once the matrix has less than 200 rows. After the exact solution is found on the coarse grid it is prolongated up one level at the time, performing a given number of (post-) smoothing steps at each level. An illustration of the V-cycle is shown in Figure 4.4a.

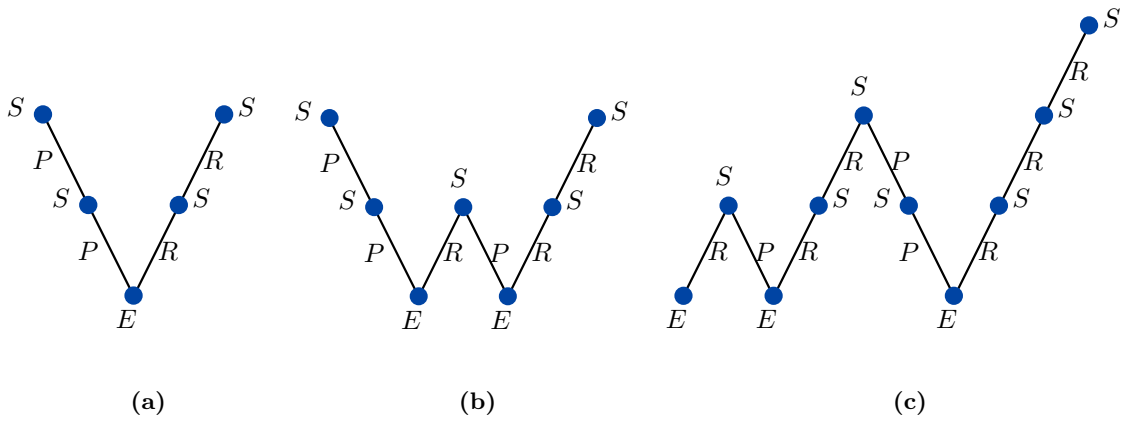


Figure 4.4: Illustration of common types of multigrid cycles; (*a*) V-cycle, (*b*) W-cycle and (*c*) the F-cycle. *S* denotes smoothing, *R* restriction of residual, *E* exact solution by a direct method and *P* prolongation.

Another common multigrid version is the W-Cycle. For this two subsequent coarse grid corrections are performed at every level, i.e., after prolongating the solution of a coarse grid correction it is smoothed a few times and then another coarse grid correction is done. This is shown in Figure 4.4b. The V- and W-cycle routines are described in Algorithm 4 where they are separated by an if statement.
Another cycle type called the F-cycle uses the coarse grid to produce an initial guess instead of correcting a smoothed iterate. This is illustrated in Figure 4.4c.

Algorithm 4 V-/W-cycle

Given: P prolongation operator on each level, α number of smoothing steps.

$u =$ **MGcycle**(matrix A, r.h.s b, initial guess u_0, smoother M)
  if Coarsest level **then**
    Solve direct $u = A^{-1}b$
  else
    Pre-smooth α times using M to obtain u, starting from u_0
    Compute residual $r = Au - b$
    Restrict residual $r^C = P^T r$
    Recursive call $\tilde{e} =$ **MGcycle**($P^T AP$,r^C,0,$P^T MP$)
    Prolongate and correct $u \leftarrow u - P\tilde{e}$
    if W-cycle **then**
      Repeat 5-9
    end if
    Post-smooth α times using M updating u
  end if
  Return u

An implementation of a complete multigrid algorithm would consist of a setup phase and the solve phase. During the setup the intergrid operator P^l, coefficient matrix A^l and smoother M^l for each level l are constructed. The setup phase is often a major contributor to the time consumption of a multigrid algorithm. For this reason the multigrid method is especially effective for problems where a relatively large number of cycles are needed and the operators can be reused. For example in dynamic problems with a time iteration the operators can be reused on each time step, or when multigrid is used as a preconditioner for a Krylov subspace method the same transfer operators can be used on each Krylov iteration.

4.2 Algebraic multigrid

In many practical cases such as for unstructured grids, irregular grids or problems where no grid is explicitly determined, the selection of coarse grids can be problematic. Algebraic multigrid (AMG) is a technique that steps away from the discretization and geometrical information and attempts to apply the multigrid principals using only information available in the system matrix. This suggests AMG as a black-box solver and opens for a wider range of problems. Although AMG need not have a direct coupling to a physical discretization its construction still needs the same fundamental components as the geometric case. There must be a sequence of (virtual) grids, intergrid operators, smoothing operators, coarse grid versions of the matrix A and a solver, usually direct, for the coarsest grid.

4.2.1 Algebraic smoothness

Instead of giving the grids a physical meaning one can construct a new grid as the undirected adjacency graph of the matrix A. While the resulting graph is similar to the discretization grid it is important to keep in mind that they are not always the same. For example, using higher order discretizations or when considering the elasticity equation the nodes in this graph represent unknowns in the algebraic equation, not nodes in the original grid. The issues that arise for such cases are discussed in Section 4.3. In the following, a point refers to an unknown in the adjacency graph, not necessarily a point in the grid.
From the adjacency graph the couplings between the points can easily be visualized. The weights on the edges represent the strength of a coupling between points. This strength is determined by the coefficients of the PDE as well as physical length between nodes. This means that a stretched grid and anisotropic

coefficients of the PDE will have a similar effect on the algebraic grid.
The adjacency graph is considered as the finest grid, and coarser grids are built from this. Before coarsening techniques can be described the concept of strong coupling has to be formally defined. In order to define this we look at what properties the resulting grid should have. The fundamental property of a coarse grid is that it should be able to represent smooth error components accurately. In geometric multigrid smoothness is characterized by a low spatial frequency. However, it is not necessarily just these low spatial frequency components we want to transfer to the coarser grid. For example, with anisotropies present an error might be smooth in one direction but oscillatory in another. This error would then not appear to be geometrically smooth even though the convergence might have slowed down. As the error is not geometrically smooth a standard isotropic coarse grid would not be able to represent the error, and the performance of the coarse grid correction would be poor. For AMG methods one instead directly characterizes *algebraically smooth* errors as those vectors that are reduced slowly under relaxation, i.e.,

$$\|Ge\|_A \approx \|e\|_A, \tag{4.15}$$

where G is the iteration matrix of the smoother. Brezina et al. argue in [9] that if (4.15) is true for e then we also have

$$e^T Ae \approx 0. \tag{4.16}$$

This motivates the following definition.

Definition 1. *Two unknowns i and j are strongly coupled if, for some $\theta \in (0, 1)$*

$$|a_{ij}| \geq \theta \sqrt{a_{ii} a_{jj}}. \tag{4.17}$$

In the coarsening algorithms the set of strongly coupled points have to be traversed, sometimes more than once, for each i. As the size of θ will generally affect the cardinality of these sets it will also have an impact on the complexity of the algorithm and can lead to a slow coarsening. Note, there are also other characterizations of algebraic smoothness [41, 13], which lead to slightly different definitions of strong coupling.

4.2.2 Classical coarsening

The classical coarsening approach was introduced by Ruge and Stüben [41], and is often called RS-coarsening. The idea is to separate all points into two disjoint sets C and F. The C-points are those fine grid points that also will serve as unknowns on the coarse grid, while the F-points are the fine grid points that are discarded on the coarse grid.
The first question that arises is how the F/C-splitting should be done. As we want algebraically smooth components to appear more oscillatory on the coarse grid, a rule should be that any two C-points are not strongly coupled. This gives us the desired coarsening effect along those directions that the error is algebraically smooth. Another rule, which ensures the quality of the interpolation, is that any F-point should either be strongly coupled to a C-point or to another F-point that is strongly coupled to a C-point. This means that for any F-point some C-point exists in its "neighborhood" and it makes sense to interpolate from these. In general it is not possible to fulfill both rules simultaneously and they are therefore used as guidelines. Routines for this splitting are described in [41].
Assuming we have done a C/F-splitting, the prolongation operator has to be constructed. This defines how an F-point will be interpolated from surrounding C-points, i.e.,

$$e_i = \sum_{k \in P_i} w_{ik} e_k \quad \text{where } i \in F \text{ and } P_i \subset C. \tag{4.18}$$

For AMG we generally do not have geometric information about the grid and can not define a linear interpolation as in Figure 4.3. Instead the approximation by neighboring values (4.17) is used. There are many ways to construct this interpolation in practice as discussed in [47], depending on how the C/F-splitting is done, the connectivities defined by A and desired sparsity of the resulting Galerkin operator.

4.2.3 Coarsening by aggregation

An alternative to the classical algebraic coarsening is the aggregation-based approach [49]. The idea behind this approach is less sophisticated, compared to the classical, and in a direct comparison the classical approach will often perform better. However, the aggregation method have gained attention because of its simplicity.

In the aggregation coarsening process all points are partitioned into disjoint sets, or aggregates, where an aggregate correspond to a single unknown in the coarse grid. This means that each fine grid point will be restricted onto only one coarse grid point. In contrast, for the RS-coarsening a fine grid point that is located between two coarse grid points will be restricted onto both of these.

The simple fine-coarse relationship gives only one nonzero per row in the prolongation operator, which simplifies the intergrid transfers and the construction of the Galerkin operator, not to mention the construction of P itself. Given a decomposition into aggregates $C_1, C_2, ...C_{n_C}$ this leads to the simple formulations

$$r_j^H = \sum_{i \in C_j} r_i^h \tag{4.19}$$

and

$$e_i^h = e_j^H \quad \text{for each } i \in C_j \tag{4.20}$$

for restriction and prolongation respectively. This results in a prolongation operator of the form

$$P = \begin{pmatrix} 1 & & & \\ 1 & & & \\ \vdots & & & \\ 1 & & & \\ & 1 & & \\ & 1 & & \\ & \vdots & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \\ & & & 1 \\ & & & \vdots \\ & & & 1 \end{pmatrix}. \tag{4.21}$$

The obvious question is which fine grid points should be grouped together and how the aggregates should be formed. There exist different approaches to form these.

In [49] the aggregates are successively defined as the first encountered strongly coupled neighborhood. The strongly coupled neighborhoods are defined as sets of points where each pair in the set are strongly coupled. This results in a greedy algorithm that most likely does not aggregate all fine grid points. To handle the remaining points two additional phases are performed. In the first of these, each unaggregated point that is strongly coupled to one or more points in an existing aggregate is added to this. If this is true for more than one existing aggregate, the one to which it is strongest coupled is chosen. In some

points remain unaggregated a last phase is performed where additional aggregates are constructed from the remaining points. The resulting algorithm is described in Algorithm 5. Typically the last phase will not be performed as all points have already been aggregated. Typical 2D aggregates are shown in Figure 4.5.

Algorithm 5 Construction of aggregates

Given: Predefined neighborhoods $N_i(\theta)$ for each $i = 1, ..., N$.
Define $R = \{1, ..., N\}$. Set $j = 0$.
for $i \in R$ **do**
  if $N_i(\theta) \subset R$ **then**
    $j \leftarrow j + 1$
    $C_j \leftarrow N_i(\theta)$
    $R \leftarrow R \setminus C_j$
  end if
end for

Copy $\widetilde{C}_k \leftarrow C_k$ for $k = 1, ..., j$
for $i \in R$ **do**
  for $k \in \{1, ..., j\}$ **do**
    if $N_i(\theta) \cap \widetilde{C}_k \neq \emptyset$ **then**
      $C_k \leftarrow C_k \cup \{i\}$
      $R \leftarrow R \setminus i$
    end if
  end for
end for

while $R \neq \emptyset$ **do**
  Pick $i \in R$
  $j \leftarrow j + 1$
  $C_j \leftarrow N_i(\theta)$
  $R \leftarrow R \setminus C_j$
end while

Using this aggregation approach one does not take into account the actual strength of a coupling once a neighborhood is defined, i.e., all unknowns included in a neighborhood are considered to be coupled with equal strength. This could potentially lead to two strongly coupled points ending up in different aggregates. Ideally, one would compare each coupling so that the strongest ones could be given priority. A different aggregation method that tries to incorporate this issue is the one used in the AGMG software [39]. For this the unknowns are pairwise grouped according to their strongest coupling. As such a pairing would only reduce the system by a factor of two the algorithm performs another pass, grouping together two pairs, giving a coarse grid reduction factor of four.

4.2.4 Coarsening by smoothed aggregation

The unsmoothed aggregation coarsening essentially results in a piecewise constant interpolation of unknowns. As pointed out in [47], while this interpolation can represent a constant function exactly, it will not be able to accurately approximate all smooth functions. As an example, consider a smooth function f in one dimension. A first order finite-element space should be able to approximate this with an error $\mathcal{O}(h)$. This is shown in Figure 4.6a where the finite-element approximation almost coincide with the function f. If we construct the coarse grid by pairwise aggregation of nodes the basis functions are formed

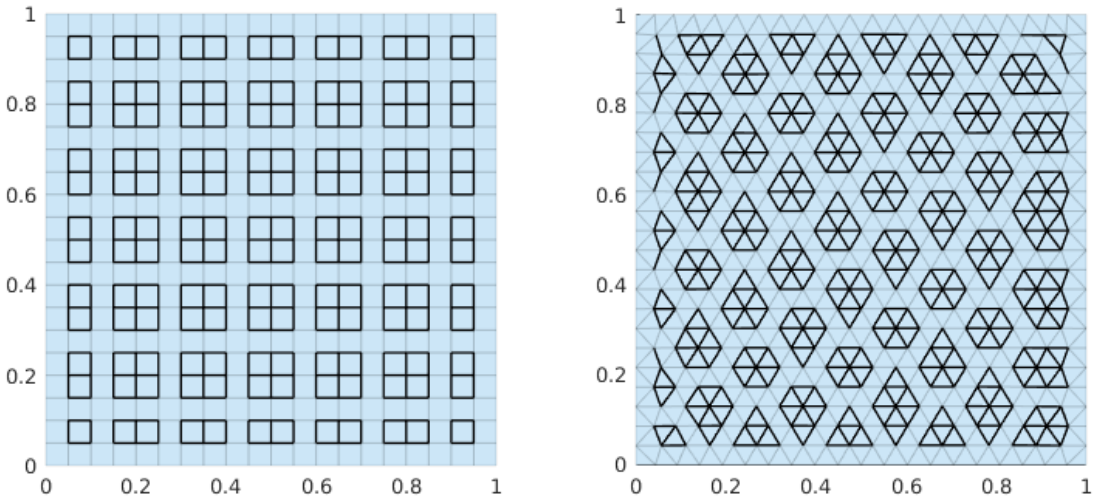


Figure 4.5: Typical result of the aggregation process described in Algorithm 5 for two different grids.

as sums of the basis functions corresponding to the two aggregated nodes. The resulting basis functions are shown at the bottom of Figure 4.6b. As the function f has a nonzero derivative at the interior of aggregates, these coarse basis functions are not able to represent f exactly. In fact, the approximation order will become zero when using an approximation based on the energy norm, as in (4.7). In Figure 4.6b u_H is the optimal approximation of f on the aggregated grid using the energy norm, which can be seen as the best approximation of the derivative at every nonempty intersection of the support of any two basis functions. For the simple equidistant example of pairwise aggregation in Figure 4.6b u_h will, with decreasing step size, look more and more like $f/2$. Braess [7] addresses this problem by multiplying the coarse grid correction with an over-relaxation factor close to two. Although this may give fairly good results in some cases, it is highly dependent on the problem at hand and generally not a robust solution.

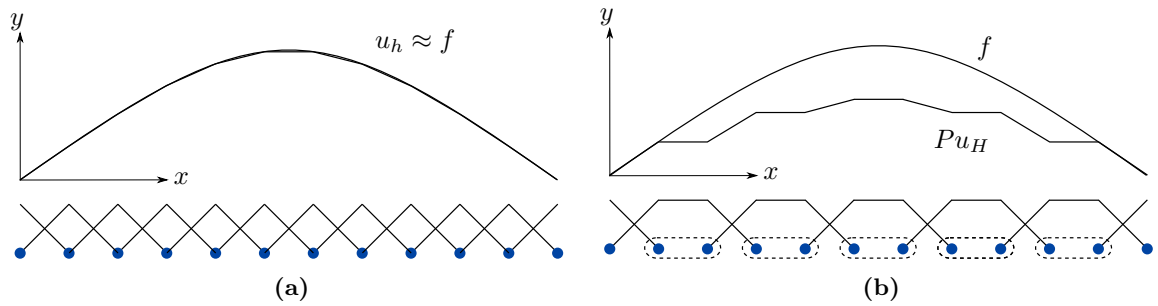


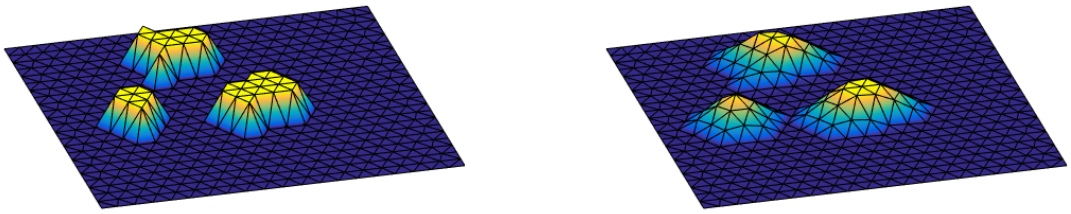
Figure 4.6: Inconsistent coarse grid approximation property for aggregation coarsening. Basis functions are shown under the plots.

The idea, as introduced in [50], to improve the simple aggregation approach above, is to use the piecewise constant interpolation only as a tentative prolongation. Subsequently, this prolongation is smoothed by applying a few iterations of a stationary iterative method. In [50] a simple ω-Jacobi relaxation step is used,

$$P = P - \omega D^{-1} A P. \tag{4.22}$$

This smoother is built on the matrix A which means that the basis functions will become locally more consistent with the PDE as they are smoothed. At the same time it smoothens out each basis function giving it a better ability to approximate smooth functions. Figure 4.7a shows an example of the tentative

basis functions on selected aggregates while Figure 4.7b shows the same basis functions after one ω-Jacobi smoothing.



(a) **(b)**

Figure 4.7: Selected basis functions for (a) unsmoothed aggregation and (b) smoothed aggregation, using ω-Jacobi smoother with $\omega = 3/2$ for a scalar problem.

As a consequence of the smoothing the support of each basis function increases. How much it increases for each smoothing iteration depends on what kind of smoother is used and how many edges that are incident on each vertex in the graph. In general for one Jacobi (or ω-Jacobi) smoothing iteration the sparsity of the resulting prolongation operator will be given from the sparsity of the squared adjacency graph of A. That is, each basis function will increase its support to include all elements that shares a boundary with the previous support. For 3D problems this will quickly increase the density of the prolongation operator, which means that the computation of the Galerkin operator will become much more expensive.

4.3 AMG for the elasticity equation

Up to now little attention has been given to the specific properties of the elasticity equation, and how to design multigrid methods for such a problem. While all the above concepts also apply in this case, elasticity introduces additional challenges.

4.3.1 Block approach for system equations

For the discretization methods described in Chapter 2, the degrees of freedom will be the displacement in each space dimension at every node. This means that the adjacency graph for the system matrix, which AMG builds upon, will be fundamentally different than the discretization grid. An aggregation-based coarsening may produce aggregates of physically incompatible degrees of freedom. This may result in deterioration of convergence. A straight forward remedy to this problem is to use a block-aggregation approach. The idea then is that if the x-displacement at two adjacent nodes are strongly coupled, then it is likely that the y-displacement at the same nodes also are. This means we perform aggregation on the discretization grid instead of the matrix graph. In order to do this we need to provide the AMG solver with information on the problem dimension and in what order the unknowns are stored. As mentioned above, for elasticity it is a convention to use an interleaved numbering of the unknowns. When building the prolongator we now insert the identity and zero block instead of one and zero, to use the same aggregation structure in all space variables. The prolongation operator would in this case typically look like the one in Figure 4.8 where two and two adjacent columns have the same structure, only shifted one row.

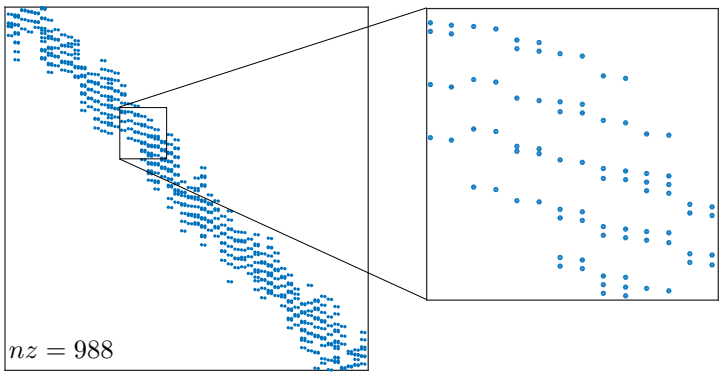


Figure 4.8: Typical prolongation operator using the block aggregation approach in 2D.

A question that arise for this block approach is how to decide whether two nodes are strongly coupled. For isotropic problems one could choose one of the space variables, discarding all the other rows and columns in A and perform aggregation as it was a scalar equation. This approach will not be justified in the case of anisotropies, where the coupling between unknowns may be different for the different space variables. In [50] this problem is addressed by defining a suitable measure of the part of A that describe the coupling between two nodes. Let $df(i)$ be the index list of unknowns associated with node i. The coupling between node i and j can now be expressed by

$$A_{(ij)} = A(df(i), df(j)). \tag{4.23}$$

Using this matrix selection we can generalize the concept of strong coupling.

Definition 2. *Two nodes i and j are strongly coupled if, for some $\epsilon \in (0, 1)$*

$$\rho\left(A_{(ii)}^{-1/2} A_{(ij)} A_{(jj)}^{-1/2}\right) > \theta. \tag{4.24}$$

This definition is the direct generalization of the scalar case (4.17). Although the size of A_{ij} is only the spatial dimension, the computation of the spectral norm ρ becomes quite costly since it has to be done for each pair of nodes in the grid. As pointed out in [23] it usually suffices to have an upper bound for the spectral norm, which means that other cheaper matrix norms can be used.

4.3.2 Interpolation of rigid body modes

As mentioned above it is important that the prolongation preserves the near-kernel elements. For a scalar problem the unsmoothed aggregation process satisfy this trivially by constructing the coarse grid basis functions as the restriction of the constant function $c(x, y) = 1$ onto each aggregate. Hence, with disjoint aggregates the basis functions will be a partition of unity. For the smoothed aggregation method, where the aggregates start to intersect, we are not guaranteed to be able to recover this partition of unity without modifying the smoothed aggregation basis functions. But for a suitable choice of prolongation smoother the smoothing should not alter this partition of unity.
For higher order problems and system PDE's the kernel contains more than just constant functions. In particular, for elasticity, the rigid body modes, consisting of the constant translation modes and linear rotation modes, defines the kernel, as discussed in Section 4.3.2.
The translational modes are taken care of in the same way as for the scalar case. The preservation of the linear rotational modes is not as simple. In Figure 4.9 the plane $z(x, y) = x$ on the coarse grid, which is in $ker(A^H)$, is prolongated onto the fine grid using an aggregation prolongator. Clearly this introduces components on the fine grid that are not in the kernel of A^h. As explained in [50] this can be corrected by introducing additional degrees of freedom on the coarse grid that are constructed to reproduce the

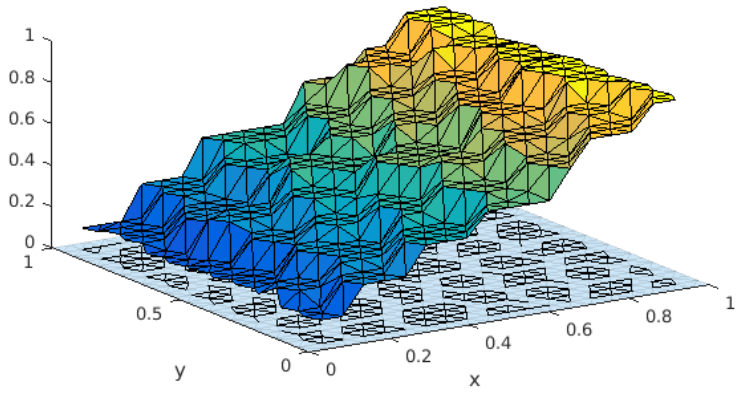


Figure 4.9: A linear function on the coarse grid prolongated to the fine grid using a piecewise constant prolongator.

linear modes. In general, we can construct a prolongator that preserves any number of kernel elements by first producing aggregates of nodes and then build coarse grid basis functions as restrictions of the kernel basis onto each aggregate. In order to use this approach the user needs to provide information about the near-kernel elements. Building the prolongator this way, these user-provided elements are often called *seed*-vectors as the prolongator is "grown" on them. For elasticity these are the rotational modes $\mathcal{R}$, discussed in Chapter 2. The prolongator is built one aggregate at the time, using the restriction of $\mathcal{R}$ onto the aggregate as columns. Given aggregates $C_1, ..., C_{n_c}$ the prolongator is formed

$$P = \begin{pmatrix} \mathcal{R}|_{C_1} & \mathcal{R}|_{C_2} & \cdots & \mathcal{R}|_{C_{n_c}} \end{pmatrix} \tag{4.25}$$

The restriction onto an aggregate $\mathcal{R}|_{C_j}$ consists of a block column vector where each block correspond to a member of the aggregate. For member i of aggregate C_j the block is (in 2D)

$$\left(\mathcal{R}|_{C_1}\right)_i = \begin{pmatrix} 1 & 0 & -(y_i - \bar{y}_j) \\ 0 & 1 & (x_i - \bar{x}_j) \end{pmatrix} \tag{4.26}$$

where (x_i, y_i) is the coordinate of node i and $(\bar{x}_j, \bar{y}_j)$ is the center of aggregate C_j. The reason why the center of the aggregate is subtracted from the rotation-column of (4.26) is to remove any translation modes from the restriction of the rotation and hence make the columns of P orthogonal. In addition we normalize each column so that we get

$$P^T P = I. \tag{4.27}$$

Doing this we make sure that the uniform equivalence of the discrete and continuous L^2-norm is satisfied.

4.4 AMG as preconditioner

AMG was originally constructed as a standalone solver. Although it has a very good theoretical performance it does have some weaknesses in practice. In many practical cases the selection of coarse grid and prolongation will not be optimal. The result of this non-optimality is that some specific components of the error are not captured well by the multigrid iterations. Improving the coarse grid and prolongation for a general set of situations have shown to be very expensive. As an alternative, if AMG is combined with a Krylov subspace method, these particular components are eliminated very efficiently in most cases [46]. This motivates the use of AMG as a preconditioner for the conjugate gradient method. As mentioned in Section 3.2.2, we can provide the preconditioner to the conjugate gradient procedure as a black-box

solver. The application of AMG as a preconditioner would then be to find the preconditioned residual as

$$z_k = \text{AMG}(r_k), \tag{4.28}$$

with AMG$(\cdot)$ being the AMG procedure set up with A as system matrix.

An issue that arise when using AMG as preconditioner for conjugate gradient is the symmetry requirement. Recall from Section 3.2.2 that the preconditioner M had to be symmetric. To construct a symmetric version of the multigrid cycle we have two choices. One way is to use a symmetric smoother such as Jacobi or SGS, for both pre- and post-smoothing. Another way is to use a non-symmetric smoother, e.g. SOR, for pre-smoothing and use its transpose, i.e., backward SOR, as a post-smoother.

Chapter 5

Implementation of AMG

As part of the work done in this project the smoothed aggregation AMG method presented above have been implemented in MATLAB. This have been done for experimental purposes and not with a goal of creating an alternative to existing, more efficient implementations. Having the procedure implemented in MATLAB provides an easy access to all components of the process, and an intuitive way of monitoring its performance when the application program is already running in MATLAB.
The presentation of the implementation here is intended to make the results in Section 6 reproducible for the reader.

The implementation consists of two main procedures, one setting up the AMG solver and the other executing the recursive multigrid cycle.
The multigrid cycle is a straight forward implementation of Algorithm 4. Note that the pre- and post-smoother should be the adjoint of each other when used as preconditioner.
The primary output from the setup phase is the prolongation operator mapping between the different levels. In addition to this, the galerkin operator P^TAP and smoothers are constructed for each level. The frame of the setup phase is run as a while-loop continuing until the number of unknowns at that level gets below a given value. For each level three separate procedures are run to construct the prolongation operator. A high-level description of the setup phase is presented in Algorithm 6.

Algorithm 6 Setup AMG

```
1 while Size of A > predefined value do
2     Find neighborhoods N_i(θ) for each node
3     Construct aggregates according to neighborhoods
4     Use aggregates to form tentative prolongator and smooth to obtain P
5     Use P to construct A and M for the next level
6 end while
```

5.1 Constructing neighborhoods

Finding the neighborhoods would be a trivial task for a scalar problem. One would simply have to symmetrically scale each element of A by forming $A_s = D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ and then remove all elements of A_s below θ, according to Definition (4.17).
For the system case, which is described in Section 4.3.1, this requires some more attention. We first

construct a scaled matrix similar to the scalar case as

$$A_s = D_B^{-\frac{1}{2}} A D_B^{-\frac{1}{2}}, \tag{5.1}$$

where D_B denotes the block diagonal of A. According to Definition 4.24 this scaled matrix should now be condensed by finding the spectral norm of each block. Motivated by [23] we use instead the cheaper Frobenius-norm

$$\|A\|_F := \left(\sum_{i=1}^{n} \sum_{j=1}^{n} |a_{ij}|^2 \right)^{\frac{1}{2}}. \tag{5.2}$$

This condensation is implemented using the matrix

$$Q = \begin{pmatrix} 1 & 1 & & & & \\ & & 1 & 1 & & \\ & & & & \ddots & \\ & & & & & 1 \quad 1 \end{pmatrix}, \tag{5.3}$$

which forms pairwise sums of rows when left-multiplied with a matrix. The condensed matrix $\widetilde{A}$ is then formed as

$$\widetilde{A} = \left(Q A_s^{\circ 2} Q^T \right)^{\circ \frac{1}{2}}, \tag{5.4}$$

where $A^{\circ 2}$ denotes entry-wise squaring.

A matrix representation of neighboring nodes is then found by dropping each value of $\widetilde{A}$ below θ. Note that the value θ is norm-dependent and we could not expect to use the exact same threshold as for the spectral norm.

In MATLAB this particular dropping step became a bottleneck of the setup phase when performed simultaneously on all edges in the graph. Instead, the dropping were done in the aggregation phase, utilizing the fact that the dropping need not be done for the nodes that are passively aggregated from being in the neighborhood of another node. As most of the nodes are aggregated passively this reduces the setup-time drastically.

5.2 Construction of aggregates

Having found the neighborhoods for each node the aggregation process can start. We have here used the original aggregation process proposed by Vanek [49], described in Algorithm 5. We will here only make a few remarks on the MATLAB implementation of the different stages of aggregation.

The first stage, where most of the aggregates are constructed, is done sequentially. This is because one, at all times, have to keep track of which nodes have been aggregated. We note that there are other alternative ways of performing the aggregation, not considered here, which are of more parallel nature [48].

The second stage, where no new aggregates are constructed, can be performed in parallel. One must, for each unaggregated node, find the existing aggregate to which it is strongest coupled to and add to that. In MATLAB this can be done simultaneously using sparse matrix operations.

The third stage where left-over nodes are collected into new aggregates, again have to be done sequentially. As most nodes are already aggregated by then, this stage is seldom time consuming.

5.3 Constructing the prolongator

Forming the tentative prolongator consists of restricting the rigid body modes onto the aggregates, as discussed in Section 4.3.2. The prolongation on the finest level is a mapping from translation-basis functions on fine grid nodes to translation- and rotation-basis functions on aggregates. This means that the first prolongator contains 2-by-3 blocks (3-by-6 for 3D) as shown in (4.26). Prolongators on coarser levels are mappings from translation- and rotation-basis functions to translation- and rotation-basis functions which means that these contain square blocks of the form (in 2D)

$$\widetilde{P}_0|_{(ij)} = \begin{pmatrix} 1 & 0 & -(y_i - \bar{y}_j) \\ 0 & 1 & (x_i - \bar{x}_j) \\ 0 & 0 & 1 \end{pmatrix}. \tag{5.5}$$

Once each block has been constructed the columns are normalized as

$$P_0 = \widetilde{P}_0 \left(\widetilde{P}_0^T \widetilde{P}_0 \right)^{-\frac{1}{2}}. \tag{5.6}$$

Note that $\widetilde{P}_0^T \widetilde{P}_0$ is a diagonal matrix, so finding the inverse square root is trivial. Having construted the tentative prolongator P_0 the final prolongator is found by applying one ω-Jacobi smoothing step according to equation (4.22).

Chapter 6

Numerical results

To investigate the different concepts described above we will in this chapter present results of some numerical experiments. The objective is to get a clearer view of what challenges arise when solving the linear elasticity equation numerically, and how the solution process can be accelerated using the concepts presented above.
The ultimate question when measuring the performance of a linear solver is the time it takes to reach a solution of desired accuracy. This, of course, depends on many different factors. For the numerical tests in this project little attention is given to optimization of the actual run-time. This is considered to be a large task involving many different disciplines in computer science and is best considered in a separate work. Instead, the focus is here put on the mathematical efficiency of a solver. As a useful measure, number of iteration is considered, which gives an indication of how much work that needs to be done. That being said, the run-time should not be completely ignored as the number of iterations could potentially be reduced to zero by putting increasingly more effort into each iteration, only to end up with a procedure that takes more time than before.

We have made use of MATLAB's implementation of the preconditioned conjugate gradient method, *pcg*. Using this, the preconditioner can be provided as either a single symmetric matrix, a split preconditioner or as a function handle that solves for the preconditioned residual.
Further we have used MATLAB's function *ichol* to construct incomplete Cholesky preconditioners. This can also be used to construct modified incomplete Cholesky preconditioners.

6.1 Test problems

6.1.1 Problem A: Isotropic square certesian grid

This test problem is included to investigate experimentally the performance of the smoothed aggregation AMG method described above. A material with Young's modulus $E = 10$ MPa and Poisson's ratio $\nu = 0.3$, occupies the unit square. The entire boundary is constrained by homogeneous Dirichlet boundary conditions, while a constant body force pointing in the negative y-direction is applied at the interior. The governing equations are discretized with the virtual element method and a cartesian grid is used.

Number of smoothing steps

We start by testing how the number of smoothing iterations α affect the solve time when AMG is used as a standalone solver. In Table 6.1 the total solve times reducing the residual norm by 10^{-8} are shown for different numbers of smoothing iterations. The different values of ω are chosen according to the discussion

in Section 3.1. As seen, the minimum solve time were found using one iteration of forward SOR with a moderate over-relaxation parameter for both pre- and post-smoothing.
For significant values of ω more smoothing iterations are required. This is related to the loss of smoothing property when over-relaxation is used. The forward-backward combination of SOR has the exact same amount of work per iteration as the forward-forward SOR, but results generally in more cycles, giving longer solve-times. It is included here because it provides the symmetry required when used as a preconditioner for conjugate gradient. Under this symmetry constraint, two smoothing iterations of forward-backward with a moderate over-relaxation gives the best solve-times. We can not, from this experiment, conclude that the same is true when used in combination with conjugate gradient, but its standalone performance gives an indication of what to expect when used as a preconditioner.

Table 6.1: Solve times for AMG as a standalone solver for varying number of smoothing iterations. A 100×100 grid is used.

Pre-/post-smoother	ω	$\alpha = 1$	$\alpha = 2$	$\alpha = 3$	$\alpha = 4$
FSOR/FSOR	1	0.52	0.57	0.65	0.75
FSOR/BSOR	1	0.57	0.59	0.68	0.77
SSOR/SSOR	1	0.67	0.93	1.15	1.27
FSOR/FSOR	1.5	0.33	0.41	0.51	0.57
FSOR/BSOR	1.5	0.54	0.47	0.61	0.60
SSOR/SSOR	1.5	0.59	0.74	0.88	1.05
FSOR/FSOR	1.75	0.52	0.43	0.43	0.49
FSOR/BSOR	1.75	0.84	0.67	0.53	0.58
SSOR/SSOR	1.75	0.75	0.76	0.86	1.04

Different prolongators

In Figure 6.1 the convergence history for conjugate gradient preconditioned with AMG using four different methods to construct the prolongator. The scalar aggregation approach ignores the block structure of A and treats it as a scalar problem, using Definition (4.17) to construct neighborhoods of unknowns. The block approach uses Definition (4.24) to construct neighborhoods of nodes. In the rotation aggregation approach the prolongator preserves the rotation as described in Section 4.3.2. For smoothed aggregation the prolongator is smoothed according to (4.22). This result clearly shows the importance of using a block approach for the elasticity equation. We can also see from this experiment that the smoothing of aggregates and preservation of rotation gives a considerable improvement in terms Krylov iterations. Similar results were found for a 3D analog to this experiment, but with an even more significant improvement when the rotation is preserved, especially when only one side were given Dirichlet boundary condition so that the solution contained a larger rotation-component.

Scalability

In Figure 6.2 the dependence of the convergence on the problem size is shown for our implementation of AMG. We have considered it as both a standalone solver and used as preconditioner, and included the IC(0) preconditioner for comparison. The AMG solver is set up with one smoothing iteration of SOR(1.3) at each level (backward post-smoothing when used as preconditioner).
When AMG is used as a standalone solver we see a moderate increase in number of iterations for increasing problem size. Applying Krylov iterations seems to both accelerate and make it more robust with respect to problem size.

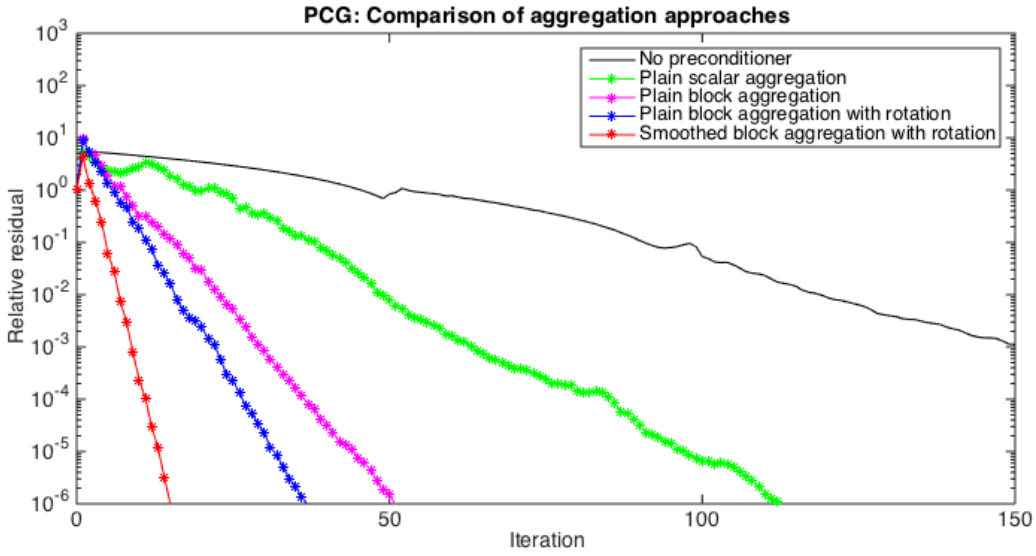


Figure 6.1: Convergence history for conjugate gradient using AMG preconditioners based on different aggregation approaches.

6.1.2 Problem B: Aspect ratio

In Problem B we consider a rectangular domain with a 100×100 grid. We let the rectangle be 1m is the y-direction and obtain different aspect ratios by varying the physical length in the x-direction from 1m to 16m. The material is isotropic with Young's modulus $E = 400$MPa and Poisson's ratio $\nu = 0.3$. In Figure 6.3 we have compared the number of iterations needed to reduce the residual norm by 10^{-8} using plain aggregation and smoothed aggregation. The blue graph shows the results using our AMG implementation. In problem A we saw that the smoothed aggregation was preferred in the isotropic case. This experiments indicates that the difference between the smoothed and unsmoothed aggregation for our implementation gets smaller as the aspect ratio increases. By investigating the aggregates we found that no couplings were dropped when forming the neighborhoods. This is not the desired behavior for large aspect ratios as nodes in the y-direction should be more strongly coupled than in the x-direction. The drop-threshold $\theta = 0.08(\frac{1}{2})^l$ thus seemed to be too low. We found, however, from experimenting that a suitable threshold, where some of the couplings are dropped and not all, was difficult to find. This could indicate that the frobenius-norm used to condense the matrix did not work well, or that the implementation were not done correctly. The loss of performance for the smoothed aggregation method could be a consequence of this. We therefore performed the same experiment using PETSc's algebraic multigrid preconditioner GAMG [4]. This uses a different aggregation method based on maximally independent sets of the squared matrix graph. In Figure 6.3 we have compared the behavior of our implementation to the GAMG preconditioner. As seen, using GAMG, the smoothed aggregation process does not loose efficiency relative to the unsmoothed aggregation. This indicates that the observed deterioration in the smoothed aggregation method relative to the unsmoothed is an artifact of our implementation.

6.1.3 Problem C: Horizontal plate

We now turn to a more challenging 3D problem. Consider a horizontal plate of thickness 0.5m, length 10m and width 5m, mounted on a wall on the shortest side. Gravity pulls the plate of density $\rho = 3000\text{kgm}^{-3}$ in the negative z-direction. The material is isotropic with Young's modulus $E = 9$GPa and Poisson's ratio $\nu = 0.3$. The grid used has 20 grid cells in all three directions. The problem is discretized using a virtual-element discretization resulting in a linear system of 26460 unknowns. Figure 6.4 shows the

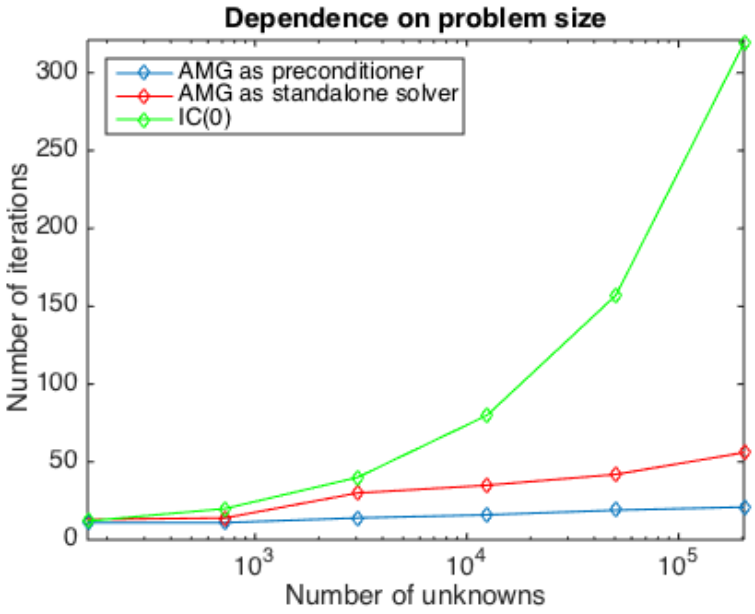


Figure 6.2: Number of iterations reducing the residual norm by 10^{-8} for different problem sizes.

divergence of the displacement on the deformed plate, which gives an indication of the change of volume in each cell. This problem is challenging because it has a thin-body feature and the grid cells have an aspect ratio of $\frac{1}{20}$. The condition number of A is estimated to 6×10^8 using MATLAB's function *condest*.

In Figure 6.5a we have considered incomplete Cholesky preconditioning, while Figure 6.5b shows the result of matrix partition based preconditioners. In Table 6.2 the solve times are presented.
We can see that the unpreconditioned conjugate gradient gradient, included in both figures for reference, uses about 3000 iterations before we see a reduction in the residual norm. This is connected to the ill-conditioning of the problem. Note that the conjugate gradient method minimizes the A-norm of the error, which monotonically decreases for each iteration. We have here plotted the 2-norm of the residual, which equals the A^2-norm of the error. This is not guaranteed to monotonically decrease. This is discussed further in [1].

Table 6.2: Solve times for one-level preconditioners on Problem C.

Preconditioner	Solve times [s]
No preconditioner	13
IC(0) shifted α	32
IC(0) shifted 0.1α	14.8
IC(0) shifted 0.01α	6
SSOR(1)	62
Jacobi	13

When constructing the IC(0) preconditioner *ichol* encountered a zero pivot, and broke down. We thus performed a shifting of the diagonal according to the discussion in Section 3.3.2. The shifting parameter α required to make A diagonally dominant was computed. As seen in Figure 6.5a, using the IC(0) preconditioner constructed from A shifted by α gave no improvement to the unpreconditioned case. By shifting only a fraction of α we saw better performances. Of course, using this partial shifting are not guaranteed to successfully construct the incomplete factors. A simple *try-catch* statement to construct the preconditioner were found useful as the iterations generally were much more time consuming than the construction of the preconditioner. The modified incomplete Cholesky MIC(0) preconditioner was also

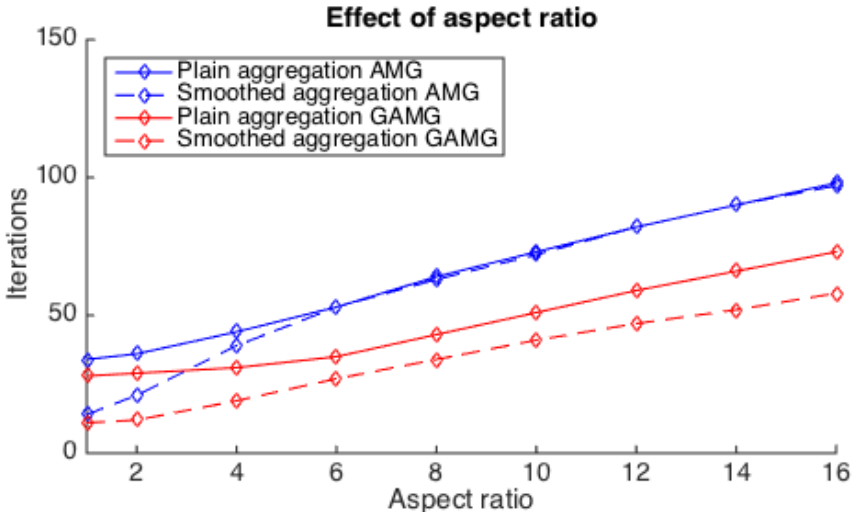


Figure 6.3: Effect of aspect ratio on smoothed and unsmoothed aggregation using our implementation (AMG) and the preconditioner GAMG in the PETSc Toolkit.

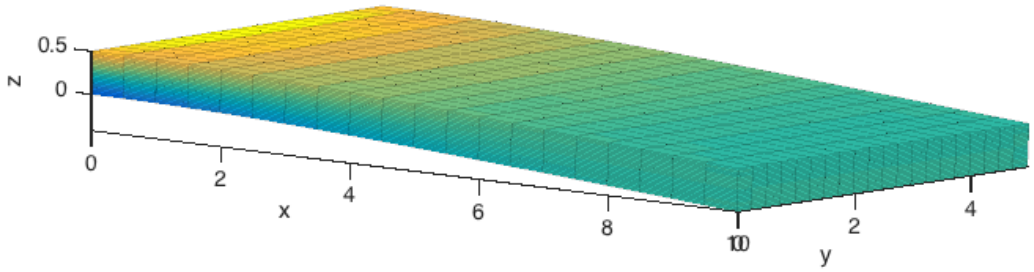
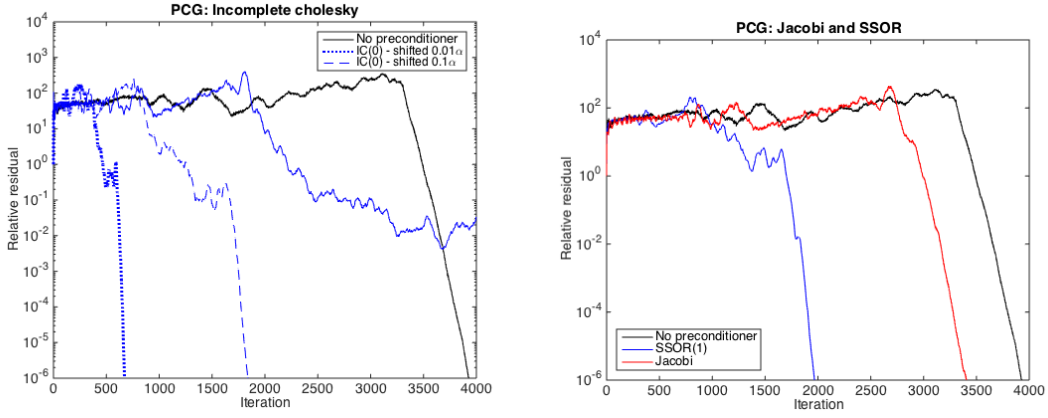


Figure 6.4: Deformed plate where the color represents the divergence of the deformation in each cell. This gives a measure of the change of volume in each cell.



(a) **(b)**

Figure 6.5: One-level preconditioners on Problem C.

considered for this experiment, but behaved similarly to IC(0), only slightly worse, and were therefore not included in the presentation here.

The SSOR preconditioner were provided to *pcg* as a split preconditioner, i.e., with $M_1 = M_{FSOR}$ and $M_2 = M_{BSOR}$. Experiments with different values of ω showed that no over-relaxation ($\omega = 1$) gave the most effective preconditioner. In fact, a value slightly less than 1 resulted in even further reduction in

number of iterations. The cost of evaluating this preconditioner was extremely high, which resulted in a total solve time more than twice the solve time for the unpreconditioned case, even though the number of iterations was nearly halved.
The Jacobi preconditioner gave a small improvement in number of iterations, but with a slightly higher cost per iteration, it ended up spending the same amount of time as the unpreconditioned iterations.

In Figure 6.6 we have presented the convergence history for the plain aggregation and the smoothed aggregation versions of our implementation of AMG. Five levels were constructed (including the original), and the setup time was 2.2s for the plain aggregation and 2.6s for the smoothed aggregation method, while the solve times were 37s and 50s, respectively. We see here the unexpected result that the plain aggregation performs significantly better than the smoothed version. We suspect that this is related to the behavior seen in Problem B for the 2D aspect ratio test. However, in Problem B the performance of the smoothed and unsmoothed method became more similar for increasing aspect ratios, while here, the smoothed performs worse. Executing the same experiment using the GAMG preconditioner in PETSc resulted in 187 and 461 iterations for the smoothed and unsmoothed method respectively. This further indicates a fault in our implementation.

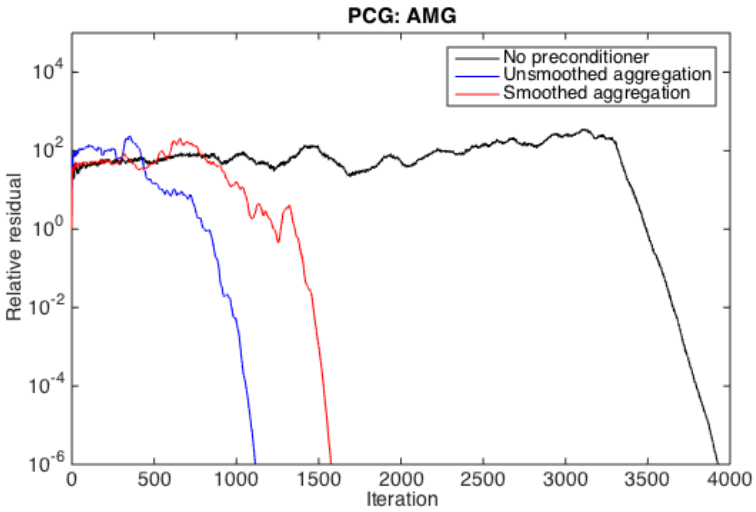


Figure 6.6: Preconditioned conjugate gradient method using the plain aggregation and the smoothed aggregation version of our AMG implementation for Problem C.

Chapter 7

Conclusion and further work

The work done in this thesis have addressed the problem of solving the elasticity equation numerically for large systems. The majority of the work concerns the use of iterative solvers for the linear system arising from discretizations.
In addition to a standard finite-element discretization for tetrahedral grids, the more recent virtual-element method have been presented, which allow the use of general polyhedral grids.
Stationary iterative methods were found to favor reduction of certain error components, resulting in a stagnation after a few iterations.
The conjugate gradient method was presented together with a theoretical result relating the convergence rate to the condition number of the system matrix.
Multigrid methods complement the stationary methods by eliminating remaining error components on coarser grids. Algebraic multigrid methods create these coarse grids automatically from information in the system matrix. When aggregation-based algebraic multigrid methods are applied to the elasticity equation a block approach should be used, aggregating grid-nodes instead of degrees of freedom. Secondly, the construction of the coarse grid should be done in such a way that the rigid body modes can be represented exactly on the coarse grid.
An implementation of the smoothed aggregation AMG method for elasticity was done in MATLAB. We saw, from numerical testing, that our implementation had a convergence rate nearly independent of the problem size when used as a preconditioner for conjugate gradient on a simple problem. Further we saw that it did not behave as expected under large aspect ratios, where the unsmoothed aggregation method performed better. This is assumed to be due to a fault in our implementation, as the GAMG preconditioner in the PETSc Toolkit did not show the same behavior.
When stationary iterations are considered as standalone solver a large value of the over-relaxation parameter gives the best convergence rate. From experiments performed in Chapter 6 we saw that when when used as smoother for a standalone AMG solver, a moderation of the over-relaxation were preferred. Further we saw that when used as preconditioner, either as smoother in the AMG-preconditioner or by itself, no over-relaxation gave the best convergence results.
The incomplete Cholesky preconditioner were found to be the most effective of the ones tested in Chapter 6. However, taking into account the scalability of our implementation seen in Section 6.1.1 and noting that our implementation is intended merely for concept studies and not suited for high-performance computing, we conclude that the smoothed aggregation AMG is indeed a competitor to IC(0).

7.1 Further work

Aggressive coarsening

The aggregation process introduced in [49], and implemented in this work, aims for a coarsening rate of 3 in each space dimension. When this method is used as a preconditioner it might be more efficient to coarsen more aggressively. For this it could be interesting to investigate the aggregation process on the square graph of A. This would enlarge the neighborhoods to include neighbors neighbors. As this would provide a larger selection of couplings it could also give more preferable aggregates in extreme anisotropic cases.

Near-incompressible materials

Near-incompressible materials, where ν is close to 0.5, are a well-known difficulty for both discretization methods and linear solvers. In further work it could therefore be interesting to consider a mixed finite-element method, such as [29], which are known to perform better in such cases. This would result in a 2-by-2 block matrix known as a saddle point problem. Appropriate preconditioners would be built on the same block structure. For this we point to the work of Rusten and Winther [42] and Mardal and Winther [36].

Complex grids

Throughout this thesis we have only considered relatively regular grids. The cases where acceleration of the linear solver is most needed are typically of more complex grids and geometric structures. In order to make use of multigrid preconditioners in such cases we must investigate its robustness with respect to the complexity of the problem. To this end we should make use of more developed implementations, such as PETSc's GAMG [4], Trilinos' ML [20] or Hypre's BoomerAMG [52], to avoid artifacts as those encountered in Section 6.1.1.

Linear solvers for the virtual-element method

We have during this thesis left out the additional part S^K of the system matrix when using the virtual-element method compared to finite-element. It would be of interest to investigate whether this additional term plays a role on the properties of the linear system. In Section 2.4 we mentioned that there was flexibility in the choice of $s^K(\cdot,\cdot)$ leading to the matrix S^K. As suggested in [19] this flexibility could potentially be taken advantage of to improve the performance of linear solvers. This is discussed further in [35].

Bibliography

[1] Mark Adams. Evaluation of three unstructured multigrid methods on 3d finite element problems in solid mechanics. *International Journal for Numerical Methods in Engineering*, 55(5):519–534, 2002.

[2] Owe Axelsson. A class of iterative methods for finite element equations. *Computer Methods in Applied Mechanics and Engineering*, 9(2):123–137, 1976.

[3] Ivo Babuska and Abdul K Aziz. Lectures on mathematical foundations of the finite element method. Technical report, MARYLAND UNIV., COLLEGE PARK. INST. FOR FLUID DYNAMICS AND APPLIED MATHEMATICS., 1972.

[4] S Balay, S Abhyankar, M Adams, J Brown, P Brune, K Buschelman, V Eijkhout, W Gropp, D Kaushik, M Knepley, et al. Petsc users manual revision 3.5. Technical report, Technical report, Argonne National Laboratory (ANL), 2014.

[5] L Beirão da Veiga, F Brezzi, A Cangiani, G Manzini, LD Marini, and A Russo. Basic principles of virtual element methods. *Mathematical Models and Methods in Applied Sciences*, 23(01):199–214, 2013.

[6] VV Belikov, VD Ivanov, VK Kontorovich, SA Korytnik, and A Yu Semenov. The non-sibsonian interpolation: A new method of interpolation of the values of a function on an arbitrary set of points. *Computational mathematics and mathematical physics*, 37(1):9–15, 1997.

[7] Dietrich Braess. Towards algebraic multigrid for elliptic problems of second order. *Computing*, 55(4):379–393, 1995.

[8] Susanne Brenner and Ridgway Scott. *The mathematical theory of finite element methods*, volume 15. Springer Science & Business Media, 2007.

[9] Marian Brezina, Andy J Cleary, Rob D Falgout, VE Henson, JE Jones, TA Manteuffel, SF McCormick, and JW Ruge. Algebraic multigrid based on element interpolation (amge). *SIAM Journal on Scientific Computing*, 22(5):1570–1592, 2001.

[10] William L Briggs, Steve F McCormick, et al. *A multigrid tutorial*. Siam, 2000.

[11] F Chalon, M Mainguy, P Longuemare, and P Lemonnier. Upscaling of elastic properties for large scale geomechanical simulations. *International journal for numerical and analytical methods in geomechanics*, 28(11):1105–1119, 2004.

[12] Tony F Chan and Howard C Elman. Fourier analysis of iterative methods for elliptic pr. *SIAM review*, 31(1):20–49, 1989.

[13] Tim Chartier, RD Falgout, VE Henson, J Jones, T Manteuffel, S McCormick, J Ruge, and PS Vassilevski. Spectral amge (ρ amge). *SIAM Journal on Scientific Computing*, 25(1):1–26, 2003.

[14] L Beirão Da Veiga, Franco Brezzi, and L Donatella Marini. Virtual elements for linear elasticity problems. *SIAM Journal on Numerical Analysis*, 51(2):794–812, 2013.

[15] James W Demmel. *Applied numerical linear algebra*, pages 279–299. Siam, 1997.

[16] James W Demmel. *Applied numerical linear algebra*, pages 307–321. Siam, 1997.

[17] SINTEF ICT Dept. of Applied Math. MATLAB Reservoir Simulation Toolbox (MRST). http://www.sintef.no/mrst. (Visited 12/14/2015).

[18] Richard S Falk. Finite element methods for linear elasticity. In *Mixed Finite Elements, Compatibility Conditions, and Applications*, pages 159–194. Springer, 2008.

[19] Arun L Gain, Cameron Talischi, and Glaucio H Paulino. On the virtual element method for three-dimensional linear elasticity problems on arbitrary polyhedral meshes. *Computer Methods in Applied Mechanics and Engineering*, 282:132–160, 2014.

[20] M.W. Gee, C.M. Siefert, J.J. Hu, R.S. Tuminaro, and M.G. Sala. ML 5.0 smoothed aggregation user's guide. Technical Report SAND2006-2649, Sandia National Laboratories, 2006.

[21] Alan George. Nested dissection of a regular finite element mesh. *SIAM Journal on Numerical Analysis*, 10(2):345–363, 1973.

[22] H Gercek. Poisson's ratio values for rocks. *International Journal of Rock Mechanics and Mining Sciences*, 44(1):1–13, 2007.

[23] Michael Griebel, Daniel Oeltz, and Marc Alexander Schweitzer. An algebraic multigrid method for linear elasticity. *SIAM Journal on Scientific Computing*, 25(2):385–407, 2003.

[24] Magnus Rudolph Hestenes and Eduard Stiefel. Methods of conjugate gradients for solving linear systems. 1952.

[25] Alan J Hoffman, Michael S Martin, and Donald J Rose. Complexity bounds for regular finite difference and finite element grids. *SIAM Journal on Numerical Analysis*, 10(2):364–369, 1973.

[26] Kai Hormann and Natarajan Sukumar. Maximum entropy coordinates for arbitrary polytopes. In *Computer Graphics Forum*, volume 27, pages 1513–1520. Wiley Online Library, 2008.

[27] Roger A Horn and Charles R Johnson. *Matrix analysis*, pages 109–111. Cambridge university press, 2012.

[28] Roger A Horn and Charles R Johnson. *Matrix analysis*. Cambridge university press, 2012.

[29] Jun Hu, Hongying Man, and Shangyou Zhang. The simplest mixed finite element method for linear elasticity in the symmetric formulation on n-rectangular grids. *arXiv preprint arXiv:1304.5428*, 2013.

[30] Ken R James. Convergence of matrix iterations subject to diagonal dominance. *SIAM Journal on Numerical Analysis*, 10(3):478–484, 1973.

[31] Tao Ju, Peter Liepa, and Joe Warren. A general geometric construction of coordinates in a convex simplicial polytope. *Computer Aided Geometric Design*, 24(3):161–178, 2007.

[32] CT Kelley. *Iterative methods for linear and nonlinear equations, SIAM, Philadelphia, 1995*, volume 65002, chapter 2.

[33] Knut-Andreas Lie. An introduction to reservoir simulation using matlab: user guide for the matlab reservoir simulation toolbox (mrst). *SINTEF ICT, Norway*, 2014.

[34] Knut-Andreas Lie, Stein Krogstad, Ingeborg Skjelkvåle Ligaarden, Jostein Roald Natvig, Halvor Møll Nilsen, and Bård Skaflestad. Open-source matlab implementation of consistent discretisations on complex grids. *Computational Geosciences*, 16(2):297–322, 2012.

[35] Konstantin Lipnikov, Gianmarco Manzini, and Mikhail Shashkov. Mimetic finite difference method. *Journal of Computational Physics*, 257:1163–1227, 2014.

[36] Kent-Andre Mardal and Ragnar Winther. Preconditioning discretizations of systems of partial differential equations. *Numerical Linear Algebra with Applications*, 18(1):1–40, 2011.

[37] Jorge Nocedal and Stephen Wright. *Numerical optimization*, chapter 5. Springer Science & Business Media, 2006.

[38] Jan Martin Nordbotten. Cell-centered finite volume discretizations for deformable porous media. *International Journal for Numerical Methods in Engineering*, 100(6):399–418, 2014.

[39] Yvan Notay. An aggregation-based algebraic multigrid method. *Electronic transactions on numerical analysis*, 37(6):123–146, 2010.

[40] Per-Olof Persson and Gilbert Strang. A simple mesh generator in matlab. *SIAM review*, 46(2):329–345, 2004.

[41] JW Ruge and Klaus Stüben. Algebraic multigrid. *Multigrid methods*, 3(13):73–130, 1987.

[42] Torgeir Rusten and Ragnar Winther. A preconditioned iterative method for saddlepoint problems. *SIAM Journal on Matrix Analysis and Applications*, 13(3):887–904, 1992.

[43] Yousef Saad. *Iterative methods for sparse linear systems*, chapter 4. Siam, 2003.

[44] Yousef Saad. *Iterative methods for sparse linear systems*, chapter 6,7. Siam, 2003.

[45] Robin Sibson. A vector identity for the dirichlet tessellation. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 87, pages 151–155. Cambridge Univ Press, 1980.

[46] K Stuben. An introduction to algebraic multigrid. *Multigrid*, pages 413–532, 2001.

[47] Klaus Stüben. A review of algebraic multigrid. *Journal of Computational and Applied Mathematics*, 128(1):281–309, 2001.

[48] Ray S Tuminaro. Parallel smoothed aggregation multigrid: Aggregation strategies on massively parallel machines. In *Proceedings of the 2000 ACM/IEEE conference on Supercomputing*, page 5. IEEE Computer Society, 2000.

[49] Petr Vanek, Jan Mandel, and Marian Brezina. *Algebraic multigrid on unstructured meshes*, volume 34. Citeseer, 1994.

[50] Petr Vaněk, Jan Mandel, and Marian Brezina. Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems. *Computing*, 56(3):179–196, 1996.

[51] Kyuichiro Washizu. *Variational methods in elasticity and plasticity*. Pergamon press, 1975.

[52] Ulrike Meier Yang et al. Boomeramg: a parallel algebraic multigrid solver and preconditioner. *Applied Numerical Mathematics*, 41(1):155–177, 2002.