TECHNISCHE UNIVERSITÄT WIEN

Standard and Auxiliary-based Algebraic Multigrid Methods for elliptic PDEs

Master Thesis

Klaus Roppert

TECHNISCHE UNIVERSITÄT WIEN

Master Thesis

Standard and Auxiliary-based Algebraic Multigrid Methods for elliptic PDEs

ausgeführt zum Zwecke der Erlangung
des akademischen Grades eines Diplom-Ingenieurs
unter der Leitung von

Univ.-Prof. Dipl.-Ing. Dr.techn. Manfred Kaltenbacher
Mitwirkung Projekt.Ass. Dipl.-Ing. Stefan Schoder
Institut für Mechanik und Mechatronik, E325 A4

eingereicht an der Technischen Univeristät Wien
Fakultät für Maschinenwesen und Betriebswissenschaften
von

Klaus Roppert
Matrikelnummer 1226189
Viktor Kaplangasse 15
2603 Felixdorf

Wien, am 12.07.2017 Unterschrift

Eidesstattliche Erklärung

Ich erkläre an Eides Statt, dass ich meine Diplomarbeit mit dem Titel *Standard and Auxiliary-based Algebraic Multigrid Methods for elliptic PDEs* selbständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe und dass ich alle Stellen, die ich wörtlich oder sinngemäß aus Veröffentlichungen entnommen habe, als solche kenntlich gemacht habe. Die Arbeit hat bisher in gleicher oder ähnlicher Form oder auszugsweise noch keiner Prüfungsbehörde vorgelegen.

Wien, den 12.07.2017

(Name Nachname)

Contents

Table of Contents ii

Notation iv

1 Introduction 1
- 1.1 Multigrid Method . . . 2
- 1.2 Introduction to Algebraic Multigrid . . . 3

2 Preliminaries 5
- 2.1 Smoothing Property of Iterative Solvers . . . 6
  - 2.1.1 Damped Jacobi Method . . . 6
  - 2.1.2 Gauss Seidel Method . . . 7
  - 2.1.3 High- and Low-frequency errors . . . 9
- 2.2 Coarse Grid Correction . . . 11
- 2.3 Multigrid Methods . . . 14
  - 2.3.0.1 Two-Grid Cycle . . . 14
  - 2.3.0.2 V- and W-Cycle . . . 14

3 General Approach to Algebraic Multigrid 17
- 3.1 Smooth error in an algebraic sense . . . 18
- 3.2 Coarsening . . . 20
  - 3.2.1 Standard/Ruge-Stüben coarsening . . . 21
  - 3.2.2 Agglomeration . . . 22
- 3.3 AMG as Preconditioner . . . 23

4 Auxiliary-based Algebraic Multigrid Methods 25
- 4.1 Auxiliary Matrix . . . 26
  - 4.1.1 Function spaces and variational formulation . . . 27
- 4.2 $H^1(\Omega)$ elliptic problems . . . 28
- 4.3 $(H^1(\Omega))^p$ elliptic problems . . . 30
  - 4.3.1 Convergence Comparisons . . . 32
- 4.4 $H(\mathrm{curl}, \Omega)$ elliptic problems . . . 33
  - 4.4.1 Convergence Comparisons . . . 35

5 Application Examples **37**
5.1 Electric potential: Capacitor . . . 37
5.1.1 Physical Description . . . 38
5.1.2 Weak Formulation . . . 38
5.1.3 Computational Setup and Results . . . 39
5.2 Mechanical Field: Loaded Beam . . . 40
5.2.1 Physical Description . . . 41
5.2.2 Weak Formulation . . . 42
5.2.3 Computational Setup and Results . . . 43
5.3 Electromagnetic Field: Current Loaded Coil . . . 44
5.3.1 Physical Description . . . 45
5.3.2 Weak Formulation . . . 46
5.3.3 Computational Setup and Results . . . 47

6 Implementation Details **50**
6.1 Sparse Matrix Storage . . . 50
6.1.1 Construction of auxiliary matrix for $H(\mathrm{curl}, \Omega)$ problems . . . 51
6.2 Smoother Algorithms . . . 53
6.2.1 Damped Jacobi . . . 53
6.2.2 Arnold-Falk-Winther . . . 54

7 Summary **57**

Literature **59**

List of Figures **61**

List of Tables **63**

List of Algorithms **64**

Notation

$\mathbb{R}$	-	Set of real numbers
$\mathbb{R}^d$	-	Set of real valued components of vector $\mathbf{x} = (x_i)_{i=1,\dots,d}$ with $x_i \in \mathbb{R}$
u	-	Scalar unknown
$\mathbf{u}$	-	Vector-valued unknown
u_h	-	Scalar unknown in the discretized system
$\underline{u}_h$	-	Vector of unknowns in the discretized system
$\mathcal{A}$	-	Bounded, linear and continuous operator
Ω	-	Computational domain
$\partial\Omega$	-	Boundary of Ω
$L^2(\Omega)$	-	Function space of square integrable functions on Ω
$(L^2(\Omega))^d$	-	Function space of d-dimensional vector-valued square integrable functions on Ω
$H^1(\Omega)$	-	Sobolev Space $H^1(\Omega) := \{u \in L^2(\Omega) : \nabla u \in (L^2(\Omega))^d\}$
$H_0^1(\Omega)$	-	$H_0^1(\Omega) := \{u \in H^1(\Omega) : u = 0 \quad \text{on} \quad \Omega\}$
$(H^1(\Omega))^d$	-	Sobolev Space $(H^1(\Omega))^d := \{\mathbf{u} \in (L^2(\Omega))^d : \nabla \cdot \mathbf{u} \in L^2(\Omega)\}$
$H(\text{curl}, \Omega)$	-	Sobolev Space $H(\text{curl}, \Omega) := \{\mathbf{u} \in (L^2(\Omega))^3 : \text{curl}(\mathbf{u}) \in (L^2(\Omega))^3\}$
$H_0(\text{curl}, \Omega)$	-	$H_0(\text{curl}, \Omega) := \{\mathbf{u} \in H(\text{curl}, \Omega) : \mathbf{u} \times \mathbf{n} = 0 \quad \text{on} \quad \partial\Omega\}$
$\mathbb{V}$	-	Function space
$\mathbb{V}^*$	-	Dual space to $\mathbb{V}$
$\mathbf{E}$	-	Electric field intensity in $\frac{V}{m}$
$\mathbf{D}$	-	Electric flux density in $\frac{As}{m^2}$
$\mathbf{H}$	-	Magnetic field intensity in $\frac{A}{m}$
$\mathbf{B}$	-	Magnetic flux density in T
$\mathbf{J}$	-	Current density in $\frac{A}{m^2}$
q_e	-	Charge density in $\frac{As}{m^3}$
V_e	-	Electric scalar potential in V
$\mathbf{v}$	-	Velocity of volume charge q_e in $\frac{m}{s}$
$[\sigma]$	-	Cauchy stress tensor with $[\sigma_{ij}] = MPa$
σ	-	Vector of Cauchy stress tensor-components (Voigt-notation)
$[\mathbf{c}]$	-	Linear strain tensor
$[\mathbf{S}]$	-	Tensor of elasticity with $[S_{ij}] = MPa$
ρ	-	Mass-density in $\frac{kg}{m^3}$
$\mathbf{f}_v$	-	Volumetric force in $\frac{N}{m^3}$
$\mathbf{a}$	-	Acceleration in $\frac{m}{s^2}$
BC	-	Boundary Condition
MG	-	Multi Grid
AMG	-	Algebraic Multi Grid
GMG	-	Geometric Multi Grid

PDE	-	Partial Differential Equation
ODE	-	Ordinary Differential Equation
CG	-	Conjugate Gradient
PCG	-	Preconditioned Conjugate Gradient
SPD	-	Symmetric Positive Definite
DOF	-	Degree of Freedom

Abstract

In the following thesis, different algebraic multigrid methods (AMG) for the solution of elliptic second order PDEs are investigated. A short introduction into multigrid methods in general is given and extended to the algebraic approach. A matlab-framework is implemented to test different algorithms and applied to various physical fields, using AMG as a standalone-solver. The most promising methods were implemented in the in-house finite element code CFS++, tested and compared to other types of iterative solvers, for three types of equations with different kernels of the underlying linear operator. The associated physical fields for the three types of equations are electrostatic, 3D-mechanic and electromagnetic. The latter one, discretized using edge-elements, the first two using nodal Lagrangian ansatz-functions.

Kurzfassung

In der folgenden Arbeit werden verschiedene Algebraische Multigrid Methoden (AMG) zur Lösung von elliptischen partiellen Differentialgleichungen zweiter Ordnung untersucht. Zu Beginn wird eine kurze, allgemeine Einführung in Multigrid Methoden gegeben, welche dann zu einem algebraischen Ansatz erweitert werden. Es wird eine Matlab-Umgebung zur Lösung dreier verschiedener physikalischer Felder entwickelt, um anschließend die vielversprechendsten Ansätze im institutseigenen Finite-Elemente Code CFS++ zu implementieren und mit anderen iterativen Gleichungslösern, anhand verschiedener Gleichungen mit unterschiedlichen Kern des linearen Operators, zu vergleichen. Mit den implementierten Lösern können elektrostatische, mechanische und elektromagnetische Probleme gelöst werden. Der Funktionenraum für die ersten beiden Felder umfasst lineare Langrange-Ansatzfunktionen, wohingegen das letztere mit Kantenelementen diskretisiert wird.

Chapter 1

Introduction

The trend of performing numerical simulations of physical problems is steadily increasing, due to less expenses in real-life experiments and shorter product development cycles. Especially when designing new products, this approach can decrease development time drastically, because different product-parameters can be tested by simply changing them automatically and re-simulate the problem, instead of performing time consuming test series or building prototypes.
In order to make the physical problem "computable", the first step of nearly all numerical methods is to discretize the continuous computational domain, for example with triangles, quadrilaterals, hexahedrals. The next step is to choose an appropriate function space for the solution and discretize this continuous function space, using *finite elements* (FE), *finite differences* (FD), *finite volumes* (FV) and many more. The majority of this thesis deals with solving linear algebraic systems, emerging from a FE-discretization of an elliptic partial differential equation (PDE) of second order, e.g. Poisson equation for electrostatic problems, Navier's equation in linear elasticity or Maxwell equations in electromagnetics. But also for other types, e.g. time-dependent hyperbolic equations or non-linear problems, linear systems, like eq.(1.1), have to be solved.

$$\mathbf{K}_h \cdot \underline{u}_h = \underline{f_h} \tag{1.1}$$

System matrices $\mathbf{K}_h \in \mathbb{R}^{N_h \times N_h}$, where N_h is the number of unknowns, are in most cases, large symmetric positive definite (SPD) matrices, which are ill conditioned (high condition number[1] $\kappa(\mathbf{K}_h)$) and depening on the discretization, sparse. Variables $\underline{u}_h$ and $\underline{f_h}$ represent the solution- and right hand side (rhs)-vector. The high demand for solving these systems present the need for a fast, accurate and memory-efficient solution process. These desired goals for the solver are, for most cases diametrically opposed, which means, we can not absolutely fulfill every parameter. Direct solvers, like the standard Gauss-elimination, Cholesky- or LU-decomposition, often fail due to the high memory consumption, which increases drastically, as the problem size increases. Especially when solving for more than one unknown per degree of freedom (DOF), e.g. three-dimensional displacements for mechanical problems, flux densities in electromagnetics and it gets even worse when various fields are coupled in so-called coupled-field-problems. Then direct solvers are not feasible anymore and iterative ones have to be used, e.g. preconditioned

[1] The condition number of the matrix $\mathbf{A} \in \mathbb{R}^{N_h \times N_h}$ is defined as the ratio between the largest and the smalles singular value of $\mathbf{A}$. For a symmetric matrix, singular values are equal to the eigenvalues.

conjugate gradient (PCG), generalized minimal residual (GMRES), biconjugate gradient (BiCG) and numerous specializations of these. Since the system matrices are in general ill-conditioned, iterative solvers can run into convergence problems and not every physical field can be solved with one particular method. In this sense, both direct and iterative solvers have their benefits but also drawbacks.
A class of very efficient iterative methods is called *multigrid methods* (MG), which are using a hierarchy of discretization spaces to filter out high- and low-frequency erros in an "optimal" way. A detailed explanation will be provided in Chapter 2.

1.1 Multigrid Method

The main idea behind all multigrid methods is to solve the problem on several FE-spaces of different size. On and between every level, different parts of the error (difference between real and numerical solution) are eliminated, respectively decreased. Since the approximation of the error on a coarser space is less complex (fewer DOF's), this method can theoretically achieve a computational complexity of $\mathcal{O}(N)$, if applied recursively on ever coarser levels, with N as the number of unknowns on the finest, original level. Early work on multigrid was done by *Hackbusch* [1], who made multigrid methods popular. Another illuminating introduction into MG methods can be found in [2], which provides deeper insight into the main step of all MG methods, the *coarse-grid-correction*, which will be explained in detail in Chapter 2. Also the transfer of the current solution and right hand side between the different hierarchies, obtained by the so-called *prolongation*- and *restriction*-operators are made plausible. From a more theoretical point of view, convergence-estimates are crucial for the consistent construction of solvers, which is also true for MG methods. In this sense, [3] provides basic introductions for convergence estimates of the *two-cycle* and *full MG-cycle*. This reference also includes information, on how to use MG methods as preconditioner for e.g. CG solvers. An important property of the system matrix, imposed implicitely or explicitely in every book or paper is that the matrix should be symmetric and positive definite (SPD). Informations on how to handle matrices which are non-SPD or not "strong enough" positive definite, are given in Chapter 4.
Until now, we have assumed that the different hierarchies of FE-spaces are already constructed and that our operators (prolongation, restriction) are defined on them. But how are those hierarchies obtained? In general there are two ways, the geometric and the algebraic approach. Coarsening an FE-space in the geometric way means, constructing different conformal[2] grid hierarchies. For simple geometries, this seems feasible but as soon as the geometry becomes more complex or the grid contains local refinements, this construction-process becomes extremely time-consuming. This is also the reason, why this geometric approach, called *geometric multigrid* (GMG) is not used in industrial applications.
The algebraic approach, called *algebraic multigrid* AMG, on the other hand, which is the main focus of this work, does not need physical grid hierarchies. It uses the underlying graph of the

[2] In an FE-context, conformal means that the traces of the ansatz-functions of two neighbouring elements must be the same at the connecting surface/edge. For example, the basis functions of element 1 are Lagrange polynomials of order 4 and element 2 consists of polynomials of order 2, then the trace of both elements does not coincide on the connecting face/edge.

system matrix to construct different FE-spaces, which can then be interpreted as fine grid nodes, elements, edges and so on.

1.2 Introduction to Algebraic Multigrid

Ruge and Stüben [4] represent the pioneering work in the field of AMG-methods. It provides a detailled (mathematically formal) introduction on smoothing, coarse grid correction and convergence estimates. A good and comprehensive definition of *algebraic smooth error* is also given but it mainly focuses on scalar nodal values. The second part of this reference focuses on the algorithmic aspect by providing different coarsening and prolongation strategies. Furthermore the necessity of requiring a M-matrix[3] is loosened, in order to obtain a robust solver. In general, the more diagonally dominant the system matrix is, the better the classical AMG algorithms work, because they represent strong connections in the FE-space, which is the starting point for coarsening algorithms. A matrix with weak diagonal dominance might induce poor convergence because the coarsening of the FE-spaces does not work well anymore. In fact a M-matrix has positive diagonal and negative off-diagonal elements and is automatically diagonal dominant but the bigger the difference between diagonal- and off-diagonal-element is, the better. The drawback of this AMG-method, compared to GMG is an increased setup time but this has to be considered in relation to the whole "coarsening process" in GMG. If we compare the time it takes to construct the physical mesh hierarchies with the setup-phase of AMG, the algebraic approach is highly beneficial in terms of performance and also more flexible.
Reitzinger [6] introduces a different, more elaborate concept for solving problems with AMG-methods, even if the system matrix has no M-matrix properties. This is done by a so-called *auxiliary-matrix*, which represents different properties of the original system matrix but has M-matrix properties and therefore we can apply normal coarsening and prolongation operators on this matrix, instead of the system matrix itself. With the help of this matrix it is also possible to extend the solution space to Nédelec's edge-elements, as presented in [7]. The term *smooth error*, in an algebraic framework, is defined less intuitive than in the geometric case, which is also described in Chapter 2, based on [6].
A further approach was introduced in [8], where a tentative prolongation operator is used to preserve the nullspace of the underlying kernel of the PDE on each coarse level, to minimize the energy in the basis functions on the coarse levels and to limit the overlap of supports of the basis functions. The last statement is equivalent to require the system matrix on the coarse levels to contain as few nonzero entries as possible. This method is called *smoothed aggregation*. It states that it has become very popular to use multigrid methods as preconditioners, rather than stand-alone solvers, mainly because of the robustness of the solution strategy. *Kaltenbacher* [9] shows an improvement of the CG method together with AMG as a preconditioner, compared to methods based on incomplete Cholesky factorization conjugate gradient methods (ICCG), because for these methods, the necessary number of iterations increases strongly with the num-

[3] A M-Matrix is defined as a matrix $\mathbf{A} = a_{ij} \in \mathbb{R}^{N_h \times N_h}$, with $a_{ij} \leq 0$ and $i \neq j$ and if it can we written as $\mathbf{A} = s\mathbf{I} - \mathbf{B}$, with $\mathbf{I}$ as the identity matrix, $\mathbf{B} = b_{ij} \geq 0$ and s larger than the largest eigenvalue of $\mathbf{B}$. Loosely speeking, this is a matrix, with positive diagonal, negative off-diagonal elements and additionally, the eigenvalues of $\mathbf{A}$ have positive real parts [5]

ber of unknowns. An introduction to the preconditioned conjugate gradient method (PCG) is also given in [9], besides the AMG-PCG algorithm and an estimation of the upper bound on the number of iterations to decrease the error to the desired level. Another source of preconditioning, using MG-methods, is [3] where the problem of ill-conditioned systems is discussed and later the construction of preconditioners is presented in a very theoretical way. Nevertheless in Section 6.3 of this reference, a "recipe" for the application of a MG-PCG method is presented.
Following the idea of [6], a virtual FE-mesh is represented by an auxiliary matrix $\mathbf{B}_h$, which needs more information of the finest grid beside the system matrix. The idea is to relate the DOF's of the system matrix to the entries of the auxiliary matrix, depending on the discretization (Lagrange or Nédélec). In the next step, the coarsening of the auxiliary matrix via an appropriate transfer operator to obtain a coarse matrix $\mathbf{B}_H$ by Galerkin's method, is applied. Again, the entries of $\mathbf{B}_H$ represent the DOF's of the original system matrix and in such a way, that the transfer operator for the system matrix is defined. As mentioned in [6], the prolongation operator has to be chosen problem dependent and in such a way, that the coarse system matrix $\mathbf{K}_H$ has the "same" properties (kernels) as the fine-grid system matrix $\mathbf{K}_h$. In previous work [10], the idea was to split an $H_0(curl, \Omega)$-function into a $(H_0^1(\Omega))^3$- and $H_0^1(\Omega)$-part and apply classical AMG for all components (Helmholtz decomposition). The newer approach coarsens the $H_0(curl, \Omega)$-matrix directly, by introducing an auxiliary matrix like before and performing a nodal-coarsening, since every entry of $\mathbf{B}_h$ corresponds to a node of the edge-element. In [7], it is shown that the coarsening can in fact be carried out on the nodes, which confirms this approach. Another feature, explained in detail in [6] is the so-called *element preconditioning*, which aims to construct an AMG preconditioner $\mathbf{C}_h$ for $\mathbf{K}_h$ from a spectrally equivalent matrix $\mathbf{B}_h$. This can be performed by applying an AMG cycle to $\mathbf{B}_h$ instead to $\mathbf{K}_h$ and numerical experiments proove the robustness and efficiency of this strategy [6].

The aim of this work is to investigate the theoretical approach to multigrid methods, especially to auxiliary-based AMG methods, proposed in [6]. Furthermore a Matlab-framework for testing different AMG methods and algorithms was developed, which recieves the linear system and some additional information as input from the in-house FEM code CFS++ [11] and computes the solution of the system. After the algorithms were verified to be working, three AMG-versions, similar to [6] were implemented into CFS++. The first type solves second order elliptic problems, stemming from a $H^1(\Omega)$ discretization, e.g. Poisson, Laplace problems. With some additional implementations (block-smoother), the second type can handle $(H^1(\Omega))^p$ discretizations of elliptic problems, e.g. Navier's equation in structural mechanics. In the third version, another type was implemented, to solve $H(curl, \Omega)$ problems from a Nédéléc edge-element discretization. All three AMG-types were applied to application examples and to verify their efficiency they are compared to standard iterative solvers.

In Chapter 2 and 3 the basics of multigrid methods in general and their specialization to AMG is presented. Followed by Chapter 4, where the ideas of auxiliary-based AMG methods from [6] and [9] are shown and the convergence results of the Matlab-implementation are presented. The application examples are then given in Chapter 5.

Chapter 2

Preliminaries

Two prominent keywords in the area of solving linear systems, which have their origins in informatics in general, are *complexity* and *scalability*. Since this work is mainly located in an engineering context, the formal definitions of both keywords from above are neglected and used in a more "heuristical" manner. The former term, also called *computational complexity*, roughly describes the number of operations needed, to solve the problem. For example, if a system with N unknowns is solved via Gauss-elimination, which has a computational complexity of about $\mathcal{O}(N^3)$, needs about N^3 operations to solve the problem. By doubling the number of unknowns $N_{\text{new}} = 2N$, the number of operations are eight times as high $\mathcal{O}(2^3N^3) = \mathcal{O}(8N^3)$. For conventional linear solvers, a lower limit in terms of complexity is $\mathcal{O}(N^2)$, which can (not formally) be made plausible, since the maximum number of matrix entries of a $N \times N$ matrix is N^2 and every entry has to be the argument of a certain binary operator. Multigrid methods on the other hand are not bound to this lower limit and can reach an optimal complexity of $\mathcal{O}(N)$ because the method transforms the solution of the $N_h \times N_h$ system to a smaller discrete space, solves it there and improves the solution by prolongating the correction back to the fine system. In an AMG framework, there are two more kinds of complexity (operator and grid), which are introduced in Chapter 3

The second term *scalability* is another important property, when describing solution methods for linear systems, especially if they are used on parallel architectures. Roughly speaking, it describes the performance benefit, when increasing the number of calculation-units. Let us assume a problem of fixed size and varying the number of calculation-nodes. If the computation time behaves approximately inversely proportional to the number of calculation-nodes, the algorithm is considered scalable. It has to be mentioned, that it is not sufficient if only the algorithm is scalable but also the implementation and hardware has to be, not going into details about the infrastructure between CPU's and GPU's or into scalable implementations using OpenMP or MPI. This behaviour is illustrated in Fig 2.1 [1]. But in fact, without algorithmic scalability, there is no parallel implementation and therefore no hardware scalability possible.

[1] `https://computation.llnl.gov/casc/sc2001_fliers/SLS/SLS01.html`

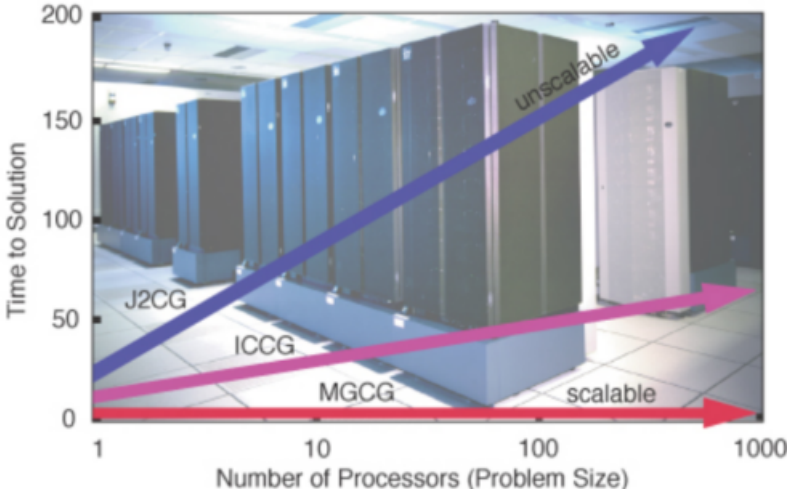

Figure 2.1: Different preconditioners for CG solver, only MG preconditioner shows scalability

2.1 Smoothing Property of Iterative Solvers

The first step of every multigrid method is the so-called *pre-smoothing*, where a simple iterative solver is applied for a few iterations (in the range of two to three), which filters out high frequency error components, as shown later on in this section. The basic principle for iterative solvers is the splitting of the system matrix $\mathbf{K}_h$ from eq.(1.1) into two, not further defined, matrices $\mathbf{K}_h = \mathbf{M} - \mathbf{N}$. Using this splitting, we obtain

$$\mathbf{M} \cdot \underline{u}_h = \mathbf{N} \cdot \underline{u}_h + \underline{b}. \tag{2.1}$$

Now we can introduce an iteration step ν and define that the left hand side is updated every iteration by evaluating the right hand side

$$\mathbf{M} \cdot \underline{u}_h^{\nu+1} = \mathbf{N} \cdot \underline{u}_h^{\nu} + \underline{b}. \tag{2.2}$$

Depending on the choice of $\mathbf{M}$ and $\mathbf{N}$, different iterative methods are obtained.

2.1.1 Damped Jacobi Method

Choosing $\mathbf{M} = \text{diag}(\mathbf{K}_h) = \mathbf{D}$, $\mathbf{N} = \mathbf{K}_h$ and introducing a damping factor ω, results in an iterative method called *damped Jacobi*

$$\underline{u}_h^{\nu+1} = \underline{u}_h^{\nu} + \omega \mathbf{D}^{-1} \left(\underline{f}_h - \mathbf{K}_h \cdot \underline{u}_h^{\nu} \right), \quad \text{for} \quad \nu = 0, 1, \ldots \quad . \tag{2.3}$$

The matrix $(\mathbf{I} - \omega \mathbf{D}^{-1} \mathbf{K}_h) := \mathbf{S}_h$ is also called smoothing operator. To show the damping property, a simple 1D Poisson problem

$$-\Delta \underline{u}_h = \underline{f}_h \qquad \text{on} \quad \Omega = (0, 1) \tag{2.4}$$

is discretized with a second order central difference scheme[2]. With a constant grid-spacing Δx, we can discretize the Laplacian operator, using $u_i = u(x_i)$, by

$$\frac{\partial^2 u}{\partial x^2}|_i \approx \frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2} \tag{2.5}$$

and the Dirichlet boundary condition translates to $u_0 = u_{N_h} = 0$. This results in a system-matrix of the form

$$\mathbf{K}_h = \frac{1}{h}\begin{pmatrix} 2 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ ... & ... & ... & ... & ... \\ 0 & 0 & 0 & -1 & 2 \end{pmatrix}. \tag{2.6}$$

Using the following initial value $\underline{u}_h^0$ of eq.(2.7) and right hand side $\underline{f}_h = \underline{0}$ (for simplicity), we can "solve" (smooth) system (2.3) with different relaxation parameters ω and study the behaviour.

$$\underline{u}_h^0 = [\text{rand}(0,1)]_{i=1}^{N_h-1} \tag{2.7}$$

For clarification, N_h represents the number of grid points[3] and in this example it was chosen to be $N_h = 100$.
In Fig.2.3 to Fig.2.8 one can clearly see the smoothing of the initally random set of values as displayed in Fig.2.2. Since the solution is zero, the plots can also be interpreted as an error[4]-plot. In the left column of the following figures, the damping of high frequency-error components works less effective than on the right side because of a non-optimal choice for the damping-parameter ω. Lowering this relaxation parameter, as it was done for Fig.2.4, 2.6 and Fig.2.8, the high frequency error decreases with fewer iterations. The damping of this presented Jacobi method can not be improved any more because the choice $\omega = \frac{2}{3}$ is already the optimal relaxation value.

2.1.2 Gauss Seidel Method

Choosing $\mathbf{M} = \mathbf{L}(\mathbf{K}_h)$ and $\mathbf{N} = \mathbf{U}(\mathbf{K}_h)$, with $\mathbf{L}$ as the lower- and $\mathbf{U}$ as the upper-diagonal matrix of $\mathbf{K}_h$, leads to the so-called *Gauss-Seidel method*. Compared to damped Jacobi, it converges about twice as fast (not shown here) but it still has a complexity of $\mathcal{O}(N_h^2)$. The iteration is then obtained by

$$\underline{u}_h^{\nu+1} = \mathbf{L}^{-1}\left(\underline{f}_h - \mathbf{U}\cdot\underline{u}_h^{\nu}\right), \quad \text{for} \quad \nu = 0, 1, ... \quad . \tag{2.8}$$

In order to show and compare the damping property to the Jacobi method, the same Laplace problem from eq.(2.4) is smoothed and the results are depicted in Fig. 2.2 to Fig. 2.8. What we can observe there, is that the solution respectively error is converging faster with the Gauss-Seidel method but the highest frequencies are damped more efficiently with the damped Jacobi method. This method can be expanded to form the *successive overrelaxation* (SOR) with the relaxation parameter ω, which improves convergence but it is not a trivial task to find an optimal

[2] A first order finite element discretization with Lagrangian ansatz-functions results in the same system matrix.
[3] later on we will use small index h for fine grid and capital index H for coarse grid
[4] *Error* in this framework means difference between numerical and analytical solution.

value for this relaxation parameter and therefore it is not used in the implemented MG methods of this thesis.

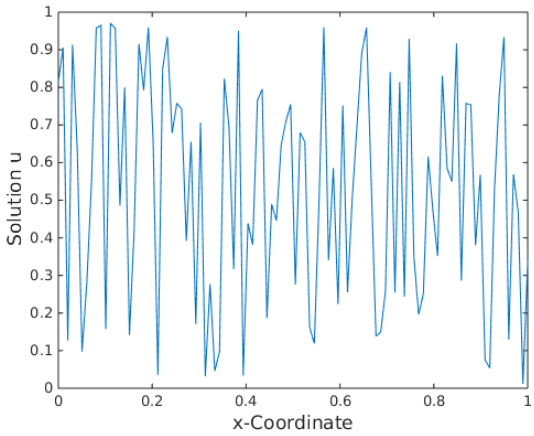

Figure 2.2: Random initial solution $\underline{u}^0$

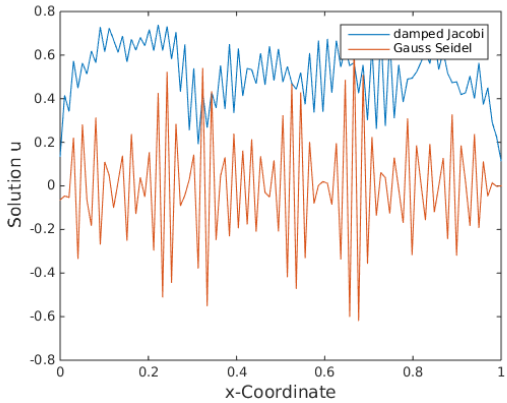

Figure 2.3: Solution $\underline{u}^5$ after 5 steps with non-optimal $\omega = 1$ for damped Jacobi

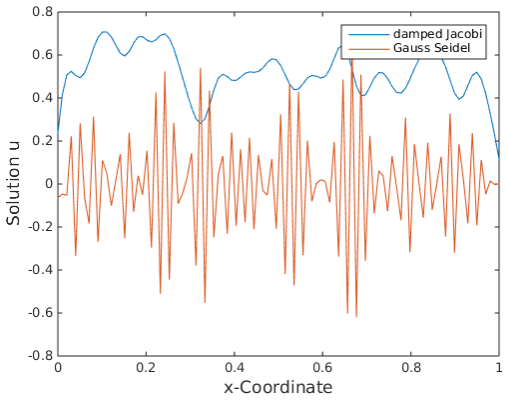

Figure 2.4: Solution $\underline{u}^5$ after 5 steps with optimal $\omega = \frac{2}{3}$ for damped Jacobi

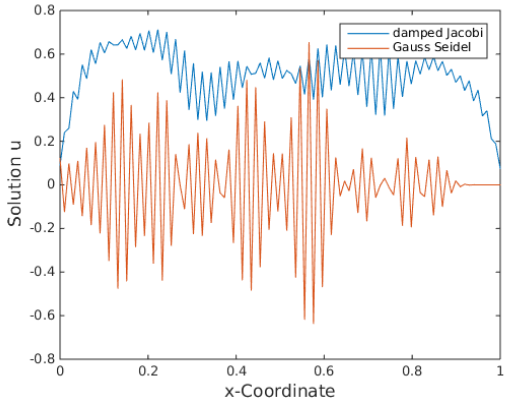

Figure 2.5: Solution $\underline{u}^5$ after 15 steps with non-optimal $\omega = 1$ for damped Jacobi

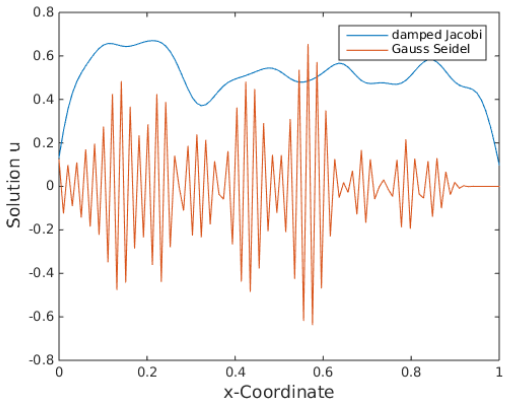

Figure 2.6: Solution $\underline{u}^5$ after 15 steps with optimal $\omega = \frac{2}{3}$ for damped Jacobi

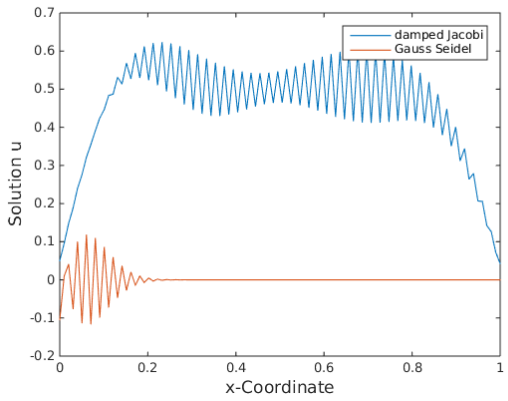

Figure 2.7: Solution $\underline{u}^5$ after 100 steps with non-optimal $\omega = 1$ for damped Jacobi

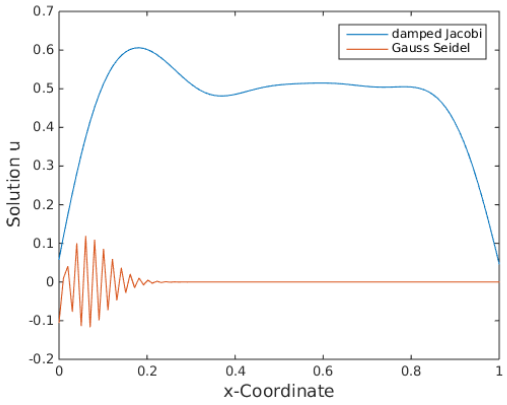

Figure 2.8: Solution $\underline{u}^5$ after 100 steps with optimal $\omega = \frac{2}{3}$ for damped Jacobi

2.1.3 High- and Low-frequency errors

When performing an eigenvalue expansion, we can describe the damping property of iterative solvers in a formal way. Therefore, we write the eigen-decomposition of the system matrix as

$$\mathbf{K}_h \cdot \underline{\Phi}_i = \lambda_i \underline{\Phi}_i, \tag{2.9}$$

with $\underline{\Phi}_i$ as the eigenvectors and λ_i as the eigenvalues of the system matrix $\mathbf{K}_h$.
In the following, the eigenvalues and eigenvectors for $\mathbf{K}_h$ are derived, since this a crucial point in understanding the smoothing property of iterative methods. Considering the entries of the system matrix with its tridiagonal structure and repeating pattern $[1, -2, 1]^T$ (stemming from the discretization using eq.(2.5)), we can express eq.(2.9) as (note that Φ_i^k is the k-th entry of the eigenvector Φ_i):

$$\frac{\Phi_i^{k-1} - 2\Phi_i^k + \Phi_i^{k+1}}{\Delta x^2} = \lambda_i \Phi_i^k. \tag{2.10}$$

For better readability, we only consider one eigenvector and eigenvalue (Φ_i^{k-1} is written as Φ_{k-1}) and rewrite eq. (2.10) by

$$\frac{\Phi_{k-1} - 2\Phi_k + \Phi_{k+1}}{\Delta x^2} = \lambda \Phi_k, \quad i = 1, ..., N_h - 1 \tag{2.11}$$

with $\Phi_0 = \Phi_{N_h+1} = 0$ due to Dirichlet boundary condition (BC). Expressing the next eigenvector-entry explicitely leads to the recurrence-like formula $\Phi_{k+1} = (2 + \lambda \Delta x^2)\Phi_k - \Phi_{k-1}$ or

$$\begin{aligned} \Phi_0(\tilde{x}) &= 0, \\ \Phi_1(\tilde{x}) &= 1, \\ \Phi_{k+1}(\tilde{x}) &= 2\tilde{x}\Phi_k(\tilde{x}) - \Phi_{k-1}(\tilde{x}), \end{aligned}$$

which might trigger ones intuition to see parallels to the recurrence formula for Chebyshev

polynomials of second kind [12]

$$\begin{aligned} U_0(x) &= 1, \\ U_1(x) &= 2x, \\ U_{k+1}(x) &= 2xU_k(x) - U_{k-1}(x). \end{aligned}$$

Due to the boundary condition $\Phi_0 = 0$ and the fact that Φ_1 is still unknown, we have to use the property that eigenvectors are only unique up to a constant factor and then we can simply scale the vector to one and shift the indices of eigenvectors by one, to coincide with the Chebyshev recurrence

$$\Phi_{k+1}(2 + \lambda\Delta x^2) = U_k(2\tilde{x}), \tag{2.12}$$

where we implicitely used the substitution $2\tilde{x} = 2 + \lambda\Delta x^2$. Due to the second Dirichlet BC $\Phi_{N_h+1} = 0$, we get the equation $U_{N_h} = 0$. Now we have reduced the problem of finding eigenvalues to finding roots of Chebyshev-polynomials, which are presented by (see e.g. in [12]):

$$\tilde{x} = \cos\left(\frac{k\pi}{N_h}\right). \tag{2.13}$$

Now we can plug the roots into the formula for the eigenvalues and obtain

$$\lambda_k = \frac{2\tilde{x} - 2}{\Delta x^2} = \frac{2\cos\left(\frac{k\pi}{N_h}\right) - 2}{\Delta x^2} = \frac{2}{\Delta x^2}\left(1 - \cos\left(\frac{k\pi}{N_h}\right)\right), \quad k = 1, ..., N_h. \tag{2.14}$$

Using a trigonometric formula, we recieve the well documented formula for the eigenvalue

$$\lambda_k = \frac{4}{\Delta x^2}\sin^2\left(\frac{k\pi}{2N_h}\right), \quad k = 1, ..., N_h. \tag{2.15}$$

For the computation of eigenvectors, we start by re-writring eq.(2.10) into the difference equation

$$\Phi_{k+1} - (2 + \lambda\Delta x^2)\Phi_k + \Phi_{k-1} = 0 \tag{2.16}$$

and define the characteristic polynomial

$$p(z) = z^2 - (2 + \lambda\Delta x^2)z + 1 = 0. \tag{2.17}$$

Since this is a polynomial of order two, the solution must be a linear combination of the roots of this polynomial. For $\lambda \in [0, 4]$, there are two conjugate complex roots and the solution must be composed as a linear combination of these. These roots can be written as

$$r_k = \cos\left(\frac{k\pi}{N_h}\right) + i\sin\left(\frac{k\pi}{N_h}\right), \tag{2.18}$$

which results in the linear combination (no complex part)

$$\Phi_k = a_1\cos\left(\frac{k\pi}{N_h}\right) + a_2\sin\left(\frac{k\pi}{N_h}\right). \tag{2.19}$$

According to the boundary conditions $\Phi_0 = \Phi_{N_h+1} = 0$, a_1 is zero and the eigenvectors follow as (in the full notation as in eq.(2.10)

$$\Phi_i^k = \sin\left(\frac{ik\pi}{N_h}\right). \tag{2.20}$$

Under the presumption that the eigenvectors form a complete function[5] system [13], every initial error can be expressed as a superposition of eigenmodes

$$\underline{e}^0 = \sum_{i=1}^{N_h} a_i \underline{\Phi}_i. \tag{2.21}$$

Inserting this result in eq. (2.3), yields[6] an error propagation of

$$\underline{e}^{(\nu+1)} = (\mathbf{I} - \omega \mathbf{D}^{-1}\mathbf{K}_h)\underline{e}^{(\nu)} = (\mathbf{I} - \omega \mathbf{D}^{-1}\mathbf{K}_h)^\nu \sum_{i=1}^{N_h} a_i \underline{\Phi}_i = \sum_{i=1}^{N_h} a_i \left(1 - \frac{h}{2}\lambda_i\right)^\nu \underline{\Phi}_i$$

$$\underline{e}^{(\nu+1)} = \sum_{i=1}^{N_h} a_i \left(1 - 2\omega \sin^2\left(\frac{i\pi}{2n}\right)\right)^\nu \underline{\Phi}_i. \tag{2.22}$$

Now, we estimate the upper bound of the embraced term for different "frequency components":

- $i = 1, ..., \frac{n}{2}$ representing the low frequency error components:

$$\left|1 - 2\omega \sin^2\left(\frac{i\pi}{2n}\right)\right| \leq \max\left\{\left|1 - 2\omega \sin^2\left(\frac{\pi}{2n}\right)\right|, |1-\omega|\right\} = \mathcal{O}(1 - \omega\frac{\pi^2}{2}h^2) \approx 1 \tag{2.23}$$

- $i = \frac{n}{2}, ..., n$ representing the high frequency error components:

$$\left|1 - 2\omega \sin^2\left(\frac{i\pi}{2n}\right)\right| \leq \max\left\{|1-\omega|, |1-2\omega|\right\} \tag{2.24}$$

It is obvious that applying the iteration ν-times, the estimates are also taken ν-times and that the low frequency components (approximated by one) stay nearly the same or at least do not grow. But the high frequency component $(1 - 2\omega)$, which is smaller one, decreases as it gets multiplied by itself.

2.2 Coarse Grid Correction

As we have seen in the last section, the high-frequency error is damped by applying some iterative smoothing steps but the low-frequency error stays nearly the same. At first glance, this might

[5] a function (vector-) system $\underline{\Phi}_1, \underline{\Phi}_2, ..., \underline{\Phi}_{N_h}$ in vector-space $\mathcal{V}$ is called *complete* $\Leftrightarrow \forall$ vectors $\underline{w} \in \mathcal{V}$, the sequence of $\underline{s}_1, \underline{s}_2, ..., \underline{s}_m = \sum_{k=1}^m a_k \underline{\Phi}_k$ converges to $\underline{w}$ or equivalently $||\underline{w} - \underline{s}_m|| \xrightarrow{m\to\infty} \underline{0}$

[6] the system matrix resulting from a second order finite difference discretisation is $\mathbf{A} = \frac{1}{h}\begin{pmatrix} 2 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ ... & ... & ... & ... & ... \\ 0 & 0 & 0 & -1 & 2 \end{pmatrix}$, this results in an inverse diagonal entry of $\mathbf{D}^{-1} = D_{i,j} = D_{i,i} = \frac{2}{\Delta x}$

seem as a shortcoming of the previously mentioned iterative solvers but taking a coarser grid into account, it will become benefitial, which is the topic of this section.
After some smoothing steps, the error might look qualitatively like Fig.2.9. If this error gets restricted to a coarser grid (for example with number of coarse nodes $N_H = \frac{N_h}{2}$), it gets relatively more "oscillatory" or "rougher" ([6],p.41). This is because of the number of possible modes in eq.(2.22). On the fine grid there are $N_h - 1$ possible modes and on the coarse grid $N_H - 1 = \frac{N_h}{2}$. If we pick a certain mode, which might be in the middle of the spectrum of the fine grid, it is at the upper-end of the spectrum on the coarse grid. This means it will be smoothed if we apply further smoothing steps on the coarse grid, which is the idea behind coarse grid correction.

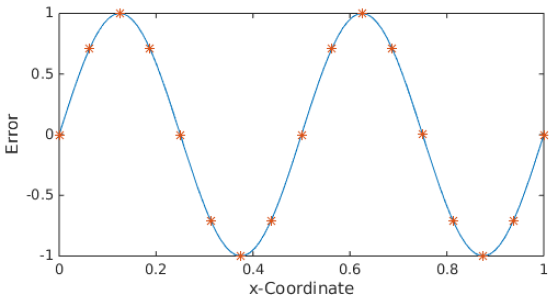

Figure 2.9: Error after some smoothing steps

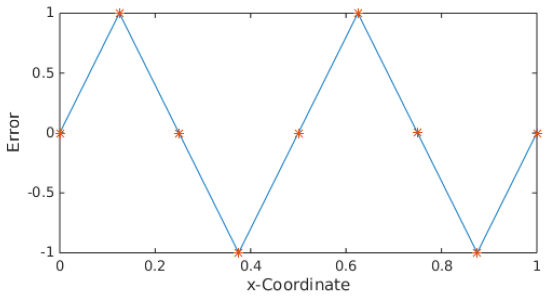

Figure 2.10: smooth error restricted to the coarse mesh

The error, described in eq.(2.25) is the difference between the numerical solution of eq.(1.1) on the fine grid $\underline{u}_h$[7] and the real/analytical one $\underline{u}$.

$$\underline{e}_h = \underline{u} - \underline{u}_h \tag{2.25}$$

Then the residual on the fine grid h can be defined as

$$\underline{r}_h = \underline{f}_h - \mathbf{K}_h \cdot \underline{u}_h = \underline{f}_h - \mathbf{K}_h \cdot (\underline{u} - \underline{e}_h)\,. \tag{2.26}$$

Inserting eq.(2.25) into eq.(2.26) yields

$$\underline{r}_h = \underline{f}_h - \mathbf{K}_h \cdot \underline{u} + \mathbf{K}_h \cdot \underline{e}_h = \mathbf{K}_h \cdot \underline{e}_h. \tag{2.27}$$

This system (2.27) is equivalent to the original discretised system $\mathbf{K}_h \cdot \underline{u}_h = \underline{f}_h$ and if the error on the fine grid $\underline{e}_h$ is smooth, then the residual on the fine grid $\underline{r}_h$ is smooth.
At this point we can show the benefit of using a coarser grid, because now we restrict the smooth residual to the coarse grid, which becomes "rough" (as shown in Fig.2.10), perform smoothening and prolongate the improved residual back to the fine grid and obtain a better approximation of the solution as with iterating on the fine grid alone. This process is called *Coarse Grid Correction (CGC)*
To write this CGC in a mathematical way, we first have to define the terms *restriction* and *prolongation*:

[7] The superscript ν, which represents the iteration number is neglected. Also instead of $\mathbf{A}$ we write $\mathbf{A}_h$ to specify that this operator acts on the fine grid with mesh-size h

- **Restriction**: This process can be thought of as a mapping from point-values on the fine grid to those on the coarse grid. This can either be done by injection (value at fine-grid point a is value at coarse-grid point b, see Fig. 2.11) or by weighting several fine-grid points to one coarse-grid point, as depicted in Fig.2.12. Written in a mathematical way, we can define a restriction operator $\mathbf{R} \in \mathbb{R}^{N_H \times N_h}$.
- **Prolongation**: This can be interpreted as the inverse operation of restriction, because it is a mapping from coarse grid to fine grid $\mathbf{R}^{-1} = \mathbf{P} \in \mathbb{R}^{N_h \times N_H}$.

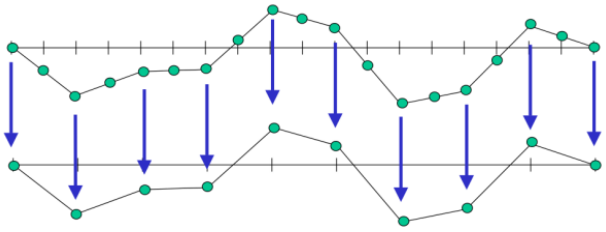

Figure 2.11: Restriction by injection [6]

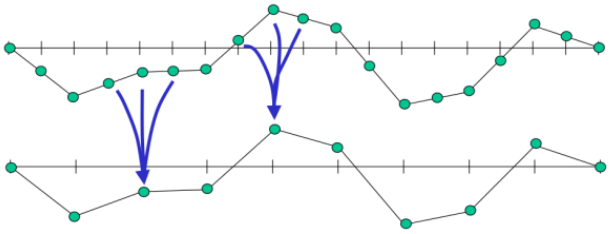

Figure 2.12: Restriction by weighting [6]

The correction can be written in pseudocode-form as shown in Algrithm 1.

Algorithm 1 Coarse Grid Correction

$\underline{u}_h^{\text{new}} \leftarrow \text{CGC}(\underline{u}_h, \underline{f}_h, \nu_1)$

function CGC
    Perform ν_1 pre-smoothing steps: $\underline{u}_h^{\nu+1} = \mathbf{S}\left(\underline{u}_h^{\nu}, \underline{f}_h\right)$
    Compute (smooth) residual on fine grid: $\underline{r_h} = \underline{f}_h - \mathbf{K}_h \cdot \underline{u}_h^{\nu_1}$
    Restrict residual to coarse grid: $\underline{r_H} = \mathbf{R} \cdot \underline{r_h}$
    Solve coarse grid problem for coarse error $\underline{e}_H$: $\mathbf{K}_H \cdot \underline{e}_H = \underline{r_H}$
    Prolongate/Interpolate the coarse-grid error to the fine grid: $\underline{e}_h = \mathbf{P} \cdot \underline{e}_H$
    Correction of the fine-grid solution: $\underline{u}_h^{\text{new}} = \underline{u}_h + \underline{e}_h$
end function

In the solving step for the coarse grid problem in Algorithm 1, the discretisation matrix $\mathbf{K}_h$ must be adapted to a coarse discretisation $\mathbf{K}_H$. The construction of coarse system matrices, preserving the operator properties is a delicate topic and will be covered in Chapter 4. Especially when

using edge-elements, the kernel of the *curl*-operator must be preserved on the coarse level.
In general, the coarse grid correction is the basis of all multigrid methods. In fact, the last ingredient, which is missing for a real multigrid method is the post-smoothing step, where the solution $\underline{u}_h^{\text{new}}$ is smoothed by some Gauss-Seidl or damped-Jacobi steps. The prolongation might trigger some higher-frequency components, which are smoothed-out to obtain a final, improved solution. In the next section, several multigrid methods are presented, which consist mainly of different concatenations of several CGC's and smoothing steps.

2.3 Multigrid Methods

Now we are equipped with the foundation to discuss different MG-methods, which are basically divided into geometrical and algebraic MG-methods. Geometric MG-methods are an extension to the methods from the last section and one can think of them as a hierarchy of real grids and not algebraic connections in the system matrix, as it is done for algebraic MG.
However, the simplicity of the geometric MG has some severe drawbacks for "real world" problems, because it is nearly impossible to provide a hierarchy of different grid-resolutions for larger, maybe even unstructured grids, as already mentioned in the introduction. Therefore an algebraic version was developed to circumvent this issue, with a convergence rate measure close to the geometric one.
In general there are four multigrid cycles (two-grid cycle, V-cycle, W-cycle and full-cycle). The algorithmic basics of these are discussed in the following.

2.3.0.1 Two-Grid Cycle

To obtain a V-cycle, a post-smoothing step in Algorithm 1 must be added, because the prolongation to the fine grid might induce some high frequency error. Actually the only purpose of this cycle, presented in Algorithm 2, is to understand the basic workflow of multigrid methods (pre-smoothing, restriction, solve coarse problem, prolongation, post-smoothing) because for large problems even the coarse problem might be too large to be solved with a fast direct solver. The idea which leads to V- or W- cycles is to shift the solving to even coarser grids than $H = \frac{h}{2}$, which will be discussed in the next section.

2.3.0.2 V- and W-Cycle

As already mentioned, the idea is to approximate the coarse grid problem by another two-grid cycle and repeat it recursively. Here, we obviously need a hierarchy of grids for different resolution levels l, as depicted in Fig.2.13.
The algorithm for V- and W-cycles is illustrated in Algorithm 3. Now we have changed the grid index h and H to the grid-level index l but between each grid level, the coarsening factor of $\frac{1}{2}$ stays the same as before. The only new variable, which is the cycle index γ, by which we can controle if it is a V- or a W-cycle (Fig.2.14)

Algorithm 2 Two Grid Cycle

$\underline{u}_h^{\text{new}} \leftarrow \text{TGC}(\underline{u}_h, \underline{f}_h, \nu_1, \nu_2)$

function TGC
  Perform ν_1 pre-smoothing steps: $\underline{u}_h^{\nu+1} = \mathbf{S}\left(\underline{u}_h^{\nu}, \underline{f}_h\right)$
  Compute (smooth) residual on fine grid: $\underline{r_h} = \underline{f}_h - \mathbf{K}_h \cdot \underline{u}_h^{\nu_1}$
  Restrict residual to coarse grid: $\underline{r_H} = \mathbf{R} \cdot \underline{r_h}$
  Solve coarse grid problem for coarse error $\underline{e}_H$: $\mathbf{K}_H \cdot \underline{e}_H = \underline{r_H}$
  Prolongate/Interpolate the coarse-grid error to the fine grid: $\underline{e}_h = \mathbf{P} \cdot \underline{e}_H$
  Correction of the fine-grid solution: $\underline{u}_h^{\text{temp}} = \underline{u}_h + \underline{e}_h$
  Perform ν_2 post-smoothing steps: $\underline{u}_h^{\text{temp},\nu_1+\nu+1} = \mathbf{S}\left(\underline{u}_h^{\text{temp},\nu_1+\nu}, \underline{f}_h\right)$
  return $\underline{u}_h^{\text{new}} = \underline{u}_h^{\text{temp},\nu_1+\nu_2}$
end function

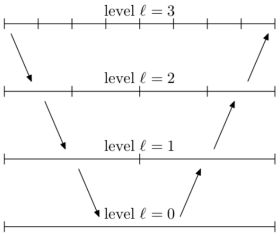

Figure 2.13: Grid hierarchy, needed for V- or W-cycles [4]

Algorithm 3 V- and W-Cycle (Multigrid Cycle)

$\underline{u}_l^{\text{new}} \leftarrow \text{MGC}(\underline{u}_l, \underline{f}_l, \nu_1, \nu_2)$

function MGC
  if $l = 0$ **then**
    Solve coarse grid problem for coarse error $\underline{e}_l$: $\mathbf{K}_l \cdot \underline{e}_l = \underline{r_l}$
  else
    Perform ν_1 pre-smoothing steps: $\underline{u}_l^{\nu+1} = \mathbf{S}\left(\underline{u}_l^{\nu}, \underline{f}_l\right)$
    Compute (smooth) residual on fine grid: $\underline{r_l} = \underline{f}_l - \mathbf{K}_l \cdot \underline{u}_l^{\nu_1}$
    Restrict residual to coarse grid: $\underline{r}_{l-1} = \mathbf{R} \cdot \underline{r_l}$
    Initialize: $\underline{e}_{l-1} = \underline{0}$
    for $i = 1 : \gamma$ **do**
      MGC$(\underline{e}_{l-1}, \underline{r}_{l-1}, \nu_1, \nu_2)$
    end for
    Prolongate/Interpolate the coarse-grid error to the fine grid: $\underline{e}_l = \mathbf{P} \cdot \underline{e}_{l-1}$
    Correction of the fine-grid solution: $\underline{u}_l^{\text{temp}} = \underline{u}_l + \underline{e}_l$
    Perform ν_2 post-smoothing steps: $\underline{u}_l^{\text{temp},\nu_1+\nu+1} = \mathbf{S}\left(\underline{u}_l^{\text{temp},\nu_1+\nu}, \underline{f}_l\right)$
  end if
  return $\underline{u}_l^{\text{new}} = \underline{u}_l^{\text{temp},\nu_1+\nu_2}$
end function

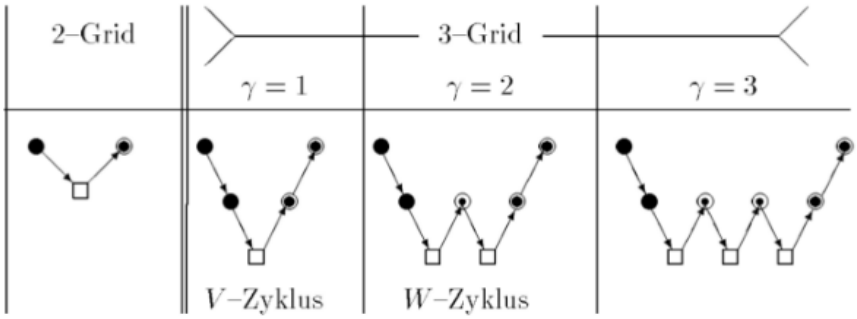

Figure 2.14: Different cycles, depending on the cycle-index γ [13]

With these multigrid-cycles, it is possible to solve large problems very efficiently, even compared to fast direct solvers. This rapid convergence and fast computation time is of course the main benefit of geometric multigrid methods but on the other hand we still have the issue of requiring a hierarchy of grids, which is the reason, why it is not widely used in industry.

Chapter 3

General Approach to Algebraic Multigrid

Another approach to obtain hierarchies of FE-spaces is to construct them, solely based on the system matrix. Methods using this way are then called algebraic multigrid methods (AMG).
The basic principles of AMG (pre-smoothing, coarse grid correction and post-smoothing) are the same as in the geometric case, only the way to obtain the hierarchies is different.
A fact, which makes AMG less efficient, compared to GMG, is the necessity of a so-called *setup phase*, which includes the construction of coarse levels and the assembling of appropriate operators, which increases the execution time. Therefore AMG is less efficient than GMG but more universally applyable. If however, we take the time, to construct different geometrical grid hierarchies, into account, the AMG may become more efficient than GMG, because the FE-space hierarchies are constructed automatically and not manually as in the geometric approach.
In the following, a short introduction into the term algebraically smooth error is given because it differs from the more intuitive geometrical interpretation. Furthermore two basic coarsening strategies are outlined, the standard Ruge-Stüben coarsening and agglomeration. Restriction- and prolongation-operators are discussed in detail, for the different AMG versions in Chapter 4. The main part of this chapter originates from [4], which is one of the most comprehensive introductions to algebraic multigrid methods.
But before we start with AMG components in particular, some basic notations are provided. Following the idea of [4], it is often easier not to think in vector-matrix terminology but in the more common grid terminology. For this purpuse, a fictitious grid/graph is introduced and the connections between grid points are identified with entries in the matrix $\mathbf{K}$. Ω^h is therefore the discrete space of matrix coefficients, corresponding to a fictitious discretised computational domain. Two points $i \in \Omega^h$ and $j \in \Omega^h$ (identified as variable u_i^h and u_j^h) are coupled if the matrix-entry $a_{ij}^h \neq 0$. Also the neighborhood of a point i can be expressed by

$$N_i^h = \left\{ j \in \Omega^h : j \neq i, a_{ij}^h \neq 0 \right\}. \tag{3.1}$$

3.1 Smooth error in an algebraic sense

In the following, the term *smooth* is defined in a mathematical way, which will be the basis of the later AMG-components.
An error can be called algebraically smooth, if it is slow to converge with respect to the smoothing operator $\mathbf{S}_h$, defined in eq.(2.3). This description of algebraic smoothness is equivalent to stating $\mathbf{S}_h \cdot \underline{e} \approx \underline{e}$, with $\underline{e}$ as the difference between numerical and real solution $\underline{e} = \underline{u} - \underline{u}_h$. For further understanding, the following norms are defined with $(\cdot, \cdot)$ as the standard Euclidean inner product:

$$||\underline{u}||_0 = (\underline{u}, \underline{u})_0^{\frac{1}{2}} = (\mathbf{D}_h \cdot \underline{u}, \underline{u})^{\frac{1}{2}}\,, \tag{3.2}$$

$$||\underline{u}||_1 = (\underline{u}, \underline{u})_1^{\frac{1}{2}} = (\mathbf{K}_h \cdot \underline{u}, \underline{u})^{\frac{1}{2}}\,, \tag{3.3}$$

$$||\underline{u}||_2 = (\underline{u}, \underline{u})_2^{\frac{1}{2}} = \left(\mathbf{D}_h^{-1}\mathbf{K}_h \cdot \underline{u}, \underline{u}\right)^{\frac{1}{2}}\,. \tag{3.4}$$

The so-called smoothing operator $\mathbf{S}_h$ can be identified (by observing the damped Jacobi method) as the following iteration matrix

$$\mathbf{S}_h = \left(\mathbf{I} - \omega \mathbf{D}_h^{-1}\mathbf{K}_h\right). \tag{3.5}$$

This iteration matrix is obtained under the assumption that the numerical solution is converging to the real solution[1] for $\nu \to \infty$. Then we can state, that eq. (2.3) also holds for the real solution $\underline{u}$ and subtracting both leads to the evolution of the error $\underline{e}^\nu = \underline{u} - \underline{u}^\nu$:

$$\mathcal{F} : \underline{u}_h^{\nu+1} = \underline{u}_h^\nu + \omega \mathbf{D}^{-1}\left(\underline{f}_h - \mathbf{K}_h \cdot \underline{u}_h^\nu\right), \quad \text{for} \quad \nu = 0, 1, ... \tag{3.6}$$

$$\mathcal{G} : \underline{u} = \underline{u} + \omega \mathbf{D}^{-1}\left(\underline{f} - \mathbf{K}_h \cdot \underline{u}\right) \tag{3.7}$$

$$\mathcal{F}\text{–}\mathcal{G} = e^{\nu+1} = (\mathbf{I} - \omega \mathbf{D}^{-1}\mathbf{K}_h)\underline{e}^\nu. \tag{3.8}$$

Then we can state the following, with $\underline{\Phi}_i$ as the eigenvectors and λ_i as the eigenvalues of $\mathbf{D}_h^{-1}\mathbf{K}_h$ (proof in [4] p.26)

$$\mathbf{D}_h^{-1}\mathbf{K}_h \cdot \underline{\Phi}_i = \lambda_i \underline{\Phi}_i \quad \rightarrow \quad ||\underline{\Phi}_i||_2^2 = \lambda_i ||\underline{\Phi}_i||_1^2, \quad ||\underline{\Phi}_i||_1^2 = \lambda ||\underline{\Phi}_i||_0^2. \tag{3.9}$$

This is of special importance because the smallest eigenvalues are those, which cause slowest convergence[2] and this is what we call *algebraic smoothness*. This definition is not the same as for the geometric one because, depending on the discretisation method it can happen, that geometrically highly oscillatory errors are algebraically smooth, see Example 3.2 in [9]. This is the reason why the term smooth should be replaced by slow-to-converge. Eq. (3.9) can also be

[1] Proof is ommited here, it is based on the spectral radius of the iteration matrix

[2] because then the smoothing-operator, respectively iteration-matrix $\mathbf{S}_h = \left(\mathbf{I} - \omega \mathbf{D}_h^{-1}\mathbf{K}_h\right)$, is close to one because (assume $\underline{e} = \underline{\Phi}$) then

$$\mathbf{S}_h \cdot \underline{e}_h = \left(\mathbf{I} - \omega \mathbf{D_h}^{-1}\mathbf{K}_h\right) \cdot \underline{e}_h$$

$$\mathbf{S}_h \cdot \underline{e}_h = \mathbf{I} \cdot \underline{e}_h - \underbrace{\omega \mathbf{D_h}^{-1}\mathbf{K}_h \underline{e}_h}_{\underbrace{\lambda_i}_{<<1}\, \underline{e}_h \omega}$$

re-written for algebraically smooth error (small λ) as

$$||\Phi||_2 \ll ||\Phi||_1 \quad \text{and} \quad ||\Phi||_1 \ll ||\Phi||_0. \tag{3.10}$$

We can then define the smoothing requirement, the smoothing operator has to satisfy by

$$||\mathbf{S}_h \cdot \underline{e}||_1^2 \leq ||\underline{e}||_1^2 - \sigma||\underline{e}||_2^2 \quad (\sigma > 0). \tag{3.11}$$

It can be shown that Gauss-Seidl and damped-Jacobi satisfy this relation uniformly. Eq.(3.11) states that $\mathbf{S}_h$ is efficient in reducing the error $\underline{e}$ as long as $||\underline{e}||_2$ is relatively large compared to $||\underline{e}||_1$, if however $||\underline{e}||_2 << ||\underline{e}||_1$ then it is inefficient and the error is called smooth.
A remarkable property of algebraic smoothness is, that the algebraic smooth error varies slowly in the direction of large connections (large off-diagonal entries a_{ij} in $\mathbf{K}$). This can be shown by using eq. (3.10) and inserting into the definition of the norms in eq. (3.4). After some algebra [4], we obtain an equivalent definition for algebraic smoothness

$$\sum_{j \neq i} \frac{|a_{ij}|}{a_{ii}} \frac{(\underline{e}_i - \underline{e}_j)^2}{\underline{e}_i^2} \ll 1. \tag{3.12}$$

Here we can see that an algebraic smooth error ($\underline{e}_i - \underline{e}_j$ is small) varies slowly in the direction of large off-diagonal connections.
This property is exploited in the coarsening step and Fig.3.1 displays, that coarsening takes place only along algebraically smooth error. The left picture shows an error-plot over the computational domain, where the lower left corner has smooth error and in the other regions we can observe heavily anisotropic errors (due to boundary conditions but this shall not be elaborated here in more detail) and the coarsening only works along paths where the error is smooth. If the error is non-smooth, respectively the connections are weak, less coarsening is applied. This coarsening process will be the topic of the next section.

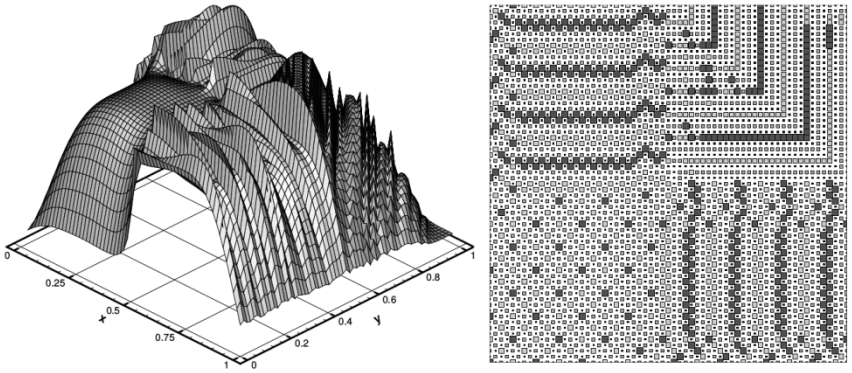

Figure 3.1: Left picture shows the error after some smoothing steps and the right one is the coarsened grid with coarsening along strong connections [14]

3.2 Coarsening

The most time-consuming part of the setup-phase in most AMG algorithms are the coarsening algorithms, which are used to subdivide the computational domain Ω into two disjoint subsets $\Omega = C \cup F$. In this context, C represents the set of nodes or entries in the system matrix, forming the coarse domain Ω^H and F represents the fine nodes or matrix-entries, which make up the fine-system, together with the coarse variables. Depending on the problem, there are numerous methods, how to obtain a useful splitting, which ensures good convergence. A good splitting does not only have to transport inhomogeneous material properties, like anisotropy, to the coarser level but also has to ensure that the underlying kernel of the fine system is still represented on the coarse grid. To transport the solution and right hand side vector of the linear system to the coarse grid, the best way for AMG is to use prolongation and restriction operators, as introduced in Section 2.2. Another possible approach is to use the equations, e.g. nodal or edge, on the coarse grid and discretize the coarse system. This approach is called coarse-grid-discretization, which is not investigated further in this work, we restrict ourselves to the first approach, using prolongation and restriction operators. To preserve the mentioned kernel of the underlying PDE of the fine system, the so-called Galerkin operator is used, which is defined as the left- and right-multiplication of the fine system $\mathbf{K}_h$ with the restriction operator $\mathbf{R}$ and interpolation (prolongation) operator $\mathbf{P}$.

$$\mathbf{K}_H = \mathbf{R}\mathbf{K}_h\mathbf{P} = \mathbf{R}\mathbf{K}_h\mathbf{R}^T. \tag{3.13}$$

At this point it should be mentioned that we assume, equivalently to geometric multigrid, that restriction and prolongation are inverse operators ($\mathbf{P}^T = \mathbf{R}$). The correct construction of these operators is essential and discussed in detail in Chapter 4. An important aspect, not only for the proper understanding of AMG but also for finite elements is the knowledge of weighted residuals. In general, we approximate a function $u(x,t)$ in space and time by

$$u(x,t) \approx u_h(x,t) = \sum_{i=1}^{N} c_j(t)\Phi_j(x), \tag{3.14}$$

where $c_j(t)$ are time-dependent coefficients and $\Phi_j(x)$ so-called ansatz-functions, e.g. Lagrangian, integrated Legendre-polynomials and many more. These ansatz-functions are elements of a function space $\mathbb{V}$ (more details about function-spaces are provided in the next chapter). Our generic differential equation, we want to solve, has the generic form

$$\mathcal{L}u = 0, \tag{3.15}$$

with the differential operator $\mathcal{L}$ and the real solution u. Applying the differential operator to the approximation of eq. (3.14), instead of the exact solution, we recieve the residual r as

$$\mathcal{L}u_h = r. \tag{3.16}$$

The aim is to minimize this residual, which is done by integrating the residual over the domain and weighing it with a so-called test-function $v(x)$:

$$\int_\Omega v_j(x) r(x,t) d\Omega = 0, \quad j = 1, ..., N. \tag{3.17}$$

Depending on the type of test-function, different methods are obtained. For example, if the domain is discretized into N disjoint subdomains Ω_i and the test-function is one if $x \in \Omega_i$ and zero elsewhere, we obtain the foundation of the finite-volume-method. By choosing the Dirac-delta function $\delta(x - x_i)$ as the test-function, a so-called collocation-method is the result. For the method we are interested in, which is the Galerkin-method, the test-functions are chosen to be identical to the ansatz-functions. Furthermore, if the function space, of which these functions are member of, is orthogonal, the residuum is projected onto the orthogonal components and the coefficients C_j can be computed.
The reason, this brief introduction was given, is the fact, that if we compute the coarse system, this orthogonalization also holds for the residual of the coarse system.

3.2.1 Standard/Ruge-Stüben coarsening

For this method, we assume a matrix with positive diagonal entries and (mostly) negative off-diagonal entries [4].
First of all we define a strong (negative) coupling of variable i to variable j by

$$\underbrace{-a_{ij}}_{\text{positive if } a_{ij}<0} \geq \epsilon_{strong} \max_{a_{ik}<0} |a_{ik}| \quad \text{with const.} \quad 0 < \epsilon_{strong} < 1. \tag{3.18}$$

Then the set of all strong couplings of variables i is defined, together with N_i from eq.(3.1), by

$$S_i = \{j \in N_i : \text{i strongly coupled to j}\}. \tag{3.19}$$

Furthermore we also need the set of all variable j from (3.19) which are strongly coupled to i and denote this set as the transpose of S_i

$$S_i^T = \{j \in \Omega : i \in S_j\}, \tag{3.20}$$

with Ω as the set of indices $(1, 2, 3, ...)$ in a_{ij}.
Now we can start with the algorithm itself. At first, every point is equipped with a "measure of importance" λ_i, which is defined as

$$\lambda_i = \left|S_i^T \cap U\right| + 2\left|S_i^T \cap F\right| \quad (i \in U), \tag{3.21}$$

with U as the set of undecided variables (neither C nor F) and F as the set of fine variables. With this λ_i we measure, roughly spoken, how valuable the variable is to become a C variable, as seen in Fig.3.2. The algorithms can be seen in Algorithm4.

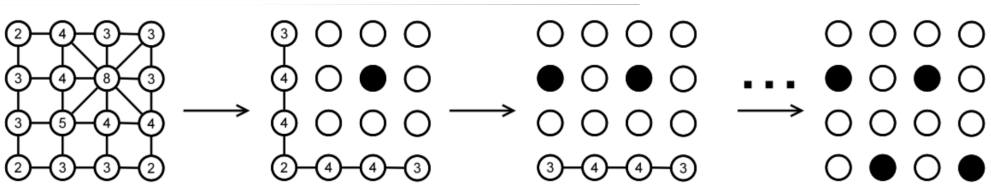

Figure 3.2: Ruge-Stüben coarsening [3], with λ_i-values for undecided points, black color for coarse- and white for fine-grid points

Algorithm 4 Standard Coarsening

1: **function** RSCoarsen
2: **while** $u \neq \{0\}$ **do**
3: Associate every point with $\lambda_i = \left|S_i^T \cap U\right| + 2\left|S_i^T \cap F\right| \quad (i \in U)$
4: Pick point with maximum value and make it a C-point $C := C \cup \{i\}, U := U \setminus i$
5: All points, strongly influenced by this new C-point become F-points
$\forall j \in S_i^T \cap U : F := F \cup \{j\}, U := U \setminus \{j\}$
6: Increase measure of newly created F-points $\lambda_k = \lambda_k + 1, \quad \forall k \in S_j$
7: **end while**
8: **return**
9: **end function**

After this algorithm, all F variables have at least one strong coupling to a C variable and the coarsening is finished (some post-coarsening steps are neglected here).

3.2.2 Agglomeration

Another coarsening algorithm, used in this work, especially later on for solving $H(\text{curl}, \Omega)$ problems, is the *agglomeration-technique*, see Fig.3.3. This strategy has less complexity than the previous RS-coarsening but also the disadvantage that the coarsening does not take place along directions of smooth error. There are different ways to obtain such a splitting. The method used in the construction of agglomerates for $H(\text{curl}, \Omega)$ problems in Chapter 4 can be describes as to define initial patches as the set of neighbours of every coarse point. After that step it is possible that some nodes have only one neighbour (there are only two nodes in the agglomerate) which can happen if Dirichlet boundary values are eliminated in the system matrix and nodes on the surface have no connection between each other. In the second step, we loop over every agglomerate, smaller than a certain threshold and redistribute them to other (neighbouring) agglomerates. By performing such an additional loop, we can bring the agglomerates, by a certain extent, to a homogeneous size of nodes. It was observed, in the $H(\text{curl}, \Omega)$-case, that such a "homogenization" of agglomerate-sizes can lead to a very good grid-complexity (GC) around 1.1 and 1.2.
This grid-complexity [9] can be used as a measure for the speed of coarsening, also for the other coarsening-strategies. It is defined as

$$GC(\mathbf{K}_h) = \frac{\sum_{i=1}^{L} M_i}{M_1}, \tag{3.22}$$

where L as the number of levels and M_i as the number of unknowns for level i. If this number is close to one, the coarsening is considered fast.

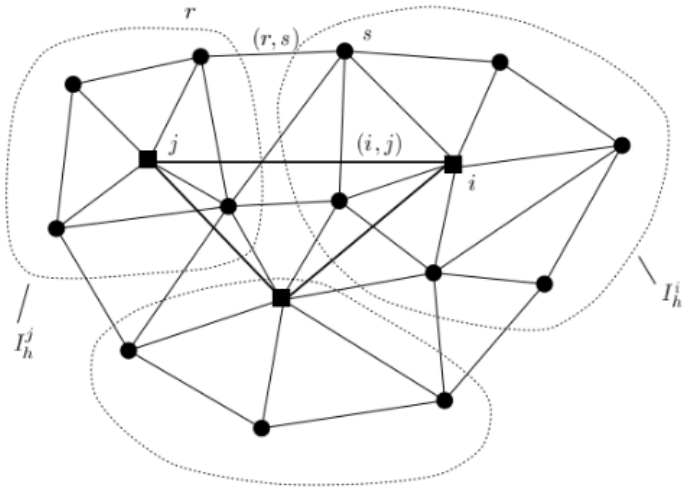

Figure 3.3: Agglomerates [6]

3.3 AMG as Preconditioner

It is shown in [15] that a robust solution strategy can be acchieved by combining CG with MG (not necessarily AMG) as a preconditioner.
By neglecting theoretical aspects of conjugate gradient method and its preconditioned version, which can be found in most books about numerical computations, the algorithm can be written (analogous to [9]), as presented in Algorithm 5, with the preconditioner-matrix $\mathbf{C}_h$.

Algorithm 5 Preconditioned Conjugate Gradient Method

function PCG
  $k = 0$
  $\underline{r}^0 = \mathbf{K}_h \underline{u}_h^0 - \underline{f}_h$
  Solve $\mathbf{C}_h \underline{d}^0 = -\underline{r}^0$
  $\underline{s}^0 = -\underline{d}^0$
  $\underline{r}^1 = \underline{r}^0$
  while $||\underline{r}^{k+1}|| > \epsilon ||\underline{r}^0||$ **do**
    $\alpha^k = \frac{(\underline{r}^k)^T \underline{s}^k}{(\underline{d}^k)^T \mathbf{K}_h \underline{d}^k}$
    $\underline{u}_h^{k+1} = \underline{u}_h^k + \alpha^k \underline{d}^k$
    $\underline{r}^{k+1} = \underline{r}^k + \alpha^k \mathbf{K}_h \underline{d}^k$
    Solve $\mathbf{C}_h \underline{s}^{k+1} = \underline{r}^{k+1}$
    $\beta^k = \frac{(\underline{r}^{k+1})^T \underline{s}^{k+1}}{(\underline{r}^k)^T \underline{s}^k}$
    $\underline{d}^{k+1} = -\underline{s}^{k+1} + \beta^k \underline{d}^k$
    $k = k + 1$
  end while
  return
end function

The optimal choice of a preconditioner matrix $\mathbf{C}_h$ would clearly be $\mathbf{C}_h = \mathbf{K}_h^{-1}$. But comput-

ing this inverse with "standard" solvers would be as demanding as solving the original problem, therefore some methods use e.g. incomplete Cholesky decomposition to obtain feasible preconditioners. An alternative, as proposed in [15] or [3] is to actually compute the inverse of $\mathbf{K}_h$ with a multigrid method, which provides a good and very fast preconditioner.
In Chapter 4, three versions of auxiliary-based AMG-methods are shown and implemented as stand-alone AMG-solvers in Matlab. In all application examples in Chapter 5, a AMG-PCG method, as proposed in Algorithm 5 is used to solve the system.

Chapter 4

Auxiliary-based Algebraic Multigrid Methods

The classic multigrid approaches, presented above are all limited to symmetric positive definite (SPD-) matrices. For scalar-valued, linear nodal Lagrangian-FE (e.g. solving the Poisson problem (2.4)) this restriction does not pose a problem. If however the unknowns are vector-valued (e.g. displacements in linear elasticity) or another function space for approximation of the continuous basis is used (e.g. $H(\mathrm{curl}, \Omega)$, the matrices are no longer positive definite or diagonally dominant. Therefore a different approach has to be used, in order to use the same algorithms from the "classic" MG-methods. This approach, as already mentioned in the introduction and described in [6], uses an *auxiliary matrix*, which represents an artificial grid and it is constructed to be diagonal dominant and SPD. The idea is to apply the coarsening not onto the system-matrix $\mathbf{K}_h$ itself, which is not nescessarily SPD, but to the auxiliary matrix $\mathbf{B}_h$, which has this property and generate coarse levels of system matrices and prolongation operators. According to [6], the information of a SPD system matrix is not enough in order to construct an efficient and robust AMG method. Therefore, additional information about the mesh-geometry, FE-discretization and the underlying PDE has to be gathered. This is the reason, why the section is called "specialized AMG-methods", because the classic AMG approach acts like a standalone-solver, which needs no further information than the sytem matrix. In the following, a short introduction to function spaces and variational formulation is given, in order to fully comprehend the construction of auxiliary matrices and prolongation operators afterwards. Further on, there are three different systems observed, similar to [6]. One arising from a scalar-valued linear nodal Lagrangian discretization, the second one from a vector-valued one and the third system, resulting from an edge-element discretization with *Nédélec*-elements. Every example of those three is solved with the implemented Matlab-AMG solver.

4.1 Auxiliary Matrix

Assume a finite element mesh ω_h, consisting of edges and nodes $\omega_h = (\omega_h^e, \omega_h^n)$, as depicted in Fig.4.1. A geometric edge is defined by two nodes $i, j \in \omega_h^n$

$$e_{ij} = (i, j) \in \omega_h^e.$$

The edge-vector (distance and direction of the nodes) can be defined by

$$\mathbf{a}_{ij} = \mathbf{x}_i - \mathbf{x}_j \in \mathbb{R}^d.$$

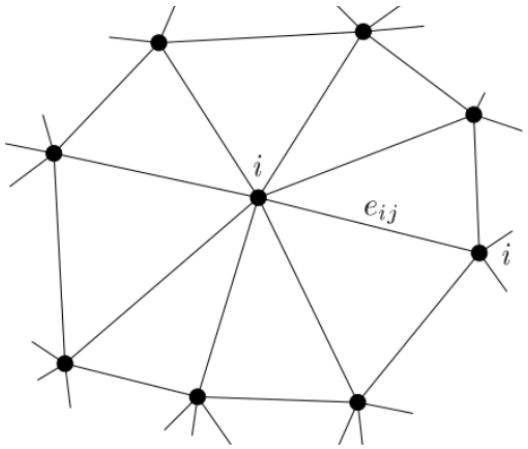

Figure 4.1: FE-mesh [6]

The auxiliary matrix $\mathbf{B}_h \in \mathbb{R}^{M_h \times M_h}$, with M_h as the number of nodes, must have the following property, in order to apply the coarsening schemes sucessively:

$$(\mathbf{B}_h)_{ij} = \begin{cases} b_{ij} \leq 0, & \text{if} \quad i \neq j \\ -\sum_{j \neq i} b_{ij} \geq 0, & \text{if} \quad i = j \end{cases} \tag{4.1}$$

In [9], the second case is defined as $1 - \sum_{j \neq i} b_{ij} \geq 0$ but for all tests, carried out, the result is the same because the off-diagonal entries are in general much larger than one. The entries of $\mathbf{B}_h$, which represents a virtual FE-mesh, have the following relation to the entries of the system matrix $\mathbf{K}_h$:

- If linear nodal Lagrange-FE are used, then $||k_{ij}|| \neq 0 \Leftrightarrow |b_{ij}| \neq 0$ for $i \neq j$
- If the system matrix stems from a scalar-valued problem, the system matrix can be used as auxiliary matrix, which results in the classic AMG method, described in the previous sections.
- If *Nédélec* edge-elements are used, b_{ij} represents an edge in the virtual FE-mesh.

4.1.1 Function spaces and variational formulation

For the theoretical background, let us introduce the operator equation, as it is done in [6]

$$\mathcal{A}u = f \quad \text{with} \quad \mathcal{A} : \mathbb{V} \to \mathbb{V}^* \tag{4.2}$$

with $\mathbb{V}$ as an appropriate function space and $\mathbb{V}^*$ its dual space[1]. Depending on the physical equation, different function spaces (Sobolev spaces) are used:

$$H^1(\Omega) = \{u \in L^2(\Omega) : \text{grad}(u) \in L^2(\Omega)\}, \tag{4.3}$$

$$(H^1(\Omega))^d := \{\mathbf{u} \in (L^2(\Omega))^d : \nabla \cdot \mathbf{u} \in L^2(\Omega)\} \tag{4.4}$$

$$H(\text{curl}, \Omega) = \{\mathbf{u} \in \left(L^2(\Omega)\right)^d : \text{curl}(\mathbf{u}) \in \left(L^2(\Omega)\right)^d\} \tag{4.5}$$

with $L^2(\Omega)$ as the space of square integrable[2] functions on Ω. If boundary conditions are introduced, the spaces from above are adapted, in order to incorporate the boundary conditions at the boundary Γ with outward pointing normal vector $\mathbf{n}$:

$$H_0^1(\Omega) = \{u \in H^1(\Omega) : u_{|\Gamma} = 0\}, \tag{4.6}$$

$$H_0(\text{curl}, \Omega) = \{\mathbf{u} \in H(\text{curl}, \Omega) : (\mathbf{u} \times \mathbf{n})_{|\Gamma} = 0\}. \tag{4.7}$$

One of the most important properties, which must be preserved across the different hierarchy-levels is the kernel (nullspace) of the linear operator $\mathcal{A}$:

$$\mathbb{V}_0 = \{u \in \mathbb{V} | a(u, v) = 0, \forall v \in \mathbb{V}\} = \ker(\mathcal{A}), \tag{4.8}$$

where $a(u, v)$ represents a bilinear form[3]. For example in electrostatics, $u \in H^1(\Omega)$ would represent the scalar electrostatic potential V_e and $v \in H^1(\Omega)$ a scalar test function. Then the bilinear form can be identified with the more convenient variational formulation-expression

$$a(V_e, v) = \int_\Omega \nabla v \cdot \nabla V_e d\Omega. \tag{4.9}$$

---

[1] *Dual space* might be unfamiliar in an engineering context, therefore the definition: Let X be a vector space, then the set of all linear functionals f on X are called the *dual space* of X. A *functional* is a map from vector space X to a scalar.

[2] Hilbert-space with $\mathbf{u}, \mathbf{v} \in \mathbb{R}^d$, dot product $\mathbf{u} \cdot \mathbf{v} = \sum_{i=1}^d u_i v_i$ and the inner product $\langle \mathbf{u}, \mathbf{v} \rangle_0 = \int_\Omega \mathbf{u} \cdot \mathbf{v} dx$ and norm $||\mathbf{u}||_0^2 = \langle \mathbf{u}, \mathbf{u} \rangle_0$. Note: x must be a measure-space, e.g. Euclidian space $\mathbb{R}^d$.

[3] *Bilinear form* on vector space $\mathbb{V}$ is a function from of two variables into the field of scalars F (of $\mathbb{V}$) $\mathbb{V} \times \mathbb{V} \to F$, which satisfy

$$\begin{gathered} B(\mathbf{v}_1 + \mathbf{v}_2, \mathbf{w}) = B(\mathbf{v}_1, \mathbf{w}) + B(\mathbf{v}_2, \mathbf{w}) \\ B(f\mathbf{v}, \mathbf{w}) = fB(\mathbf{v}, \mathbf{w}) \\ B(\mathbf{v}, \mathbf{w}_1 + \mathbf{w}_2) = B(\mathbf{v}, \mathbf{w}_1) + B(\mathbf{v}, \mathbf{w}_2) \\ B(\mathbf{v}, f\mathbf{w}) = fB(\mathbf{v}, \mathbf{w}), \end{gathered}$$

with $\mathbf{v}, \mathbf{w} \in \mathbb{V}$ and $f \in \mathbb{R}$.

In this example, the kernel of the bilinear form (equivalent to the kernel of the linear operator $\mathcal{A}$) can be identified as

$$\mathbb{V}_0 = \ker\left(a(V_e, v)\right) = \{V_e \in H^1(\Omega) | \nabla V_e = \mathbf{0}, \forall v \in H^1(\Omega)\}. \tag{4.10}$$

For a more elaborate discussion of kernel spaces, see Chapter 2 of [6].
The next step is to introduce discretized function spaces, since we are working in a FE-context. The discrete function space[4] $\mathbb{V}_h$, respectively kernel space $\mathbb{V}_{0h}$, which is discretized with a size-parameter h has to be equivalent to the original function space $\mathbb{V}$ for $h \to 0$. A prolongation- or restriction-operation from a coarse to fine system (or vice versa) is defined by

$$R_h^{\text{sys}} : \mathbb{V}_h \to \mathbb{V}_H. \tag{4.11}$$

This operation also has to preserve the kernel space on the coarse level

$$\mathbb{V}_{0h} = \{\mathbf{u} \in \mathbb{V}_h | R_h^{\text{sys}} \mathbf{u}_H \in \mathbb{V}_{0h}\}. \tag{4.12}$$

The construction of appropriate prolongation and restriction operators for three different classes of problems is presented in the following.

4.2 $H^1(\Omega)$ elliptic problems

Equipped with the definitions from above, we can now introduce the construction of AMG-components for problems in $H^1(\Omega)$ space. A possible example could be the electrostatic equation, where we are solving for a scalar potential. For a more detailed description of the physical field, see Chapter 5.

Auxiliary Matrix B_h

The auxiliary matrix can be constructed by two different methods, one is the classic AMG approach, where no additional information about the underlying PDE or geometry is needed, i.e. $\mathbf{B}_h = \mathbf{K}_h$. The other one is the geometric approach, using a different (auxiliary) matrix for the coarsening process, with

$$b_{ij} = -\frac{\epsilon}{||\mathbf{a}_{ij}||_2^2} \quad \text{for} \quad i \neq j, \tag{4.13}$$

where ϵ introduces the ability to extend the computation to more than one material, e.g. in electrostatics this could represent different permeabilities. For the diagonal, (4.1) can be used by simply summing up all off-diagonal entries of one row and multiplying it by (-1). With this construction, the matrix patterns of $\mathbf{K}_h$ and $\mathbf{B}_h$ are the same, which is important for nodal Lagrange-FE. The next step is the coarsening of the auxiliary matrix, for which the classical RS-algorithm from Section 3.2.1 is sufficient.

[4] correct partitioning of the computational domain Ω assumed

Prolongation

A very simple prolongation operator is the piecewise constant interpolation (4.14), which is the same for both $\mathbf{K}_h$ and $\mathbf{B}_h$, since both matrices have the same sparsity pattern. In [6] another prolongation operator is proposed with a discrete harmonic extension but it was neglected here because it is computationally more expensive.

$$(\mathbf{P}_h^{\text{sys}})_{ij} = (\mathbf{P}_h^B)_{ij} = \begin{cases} 1, & \text{if} \quad i = j \\ \frac{1}{|S_h^i \cap \omega_C^n|}, & \text{if} \quad i \in \omega_F^n, j \in S_h^{i,T} \cap \omega_C^n \\ 0, & \text{else} \end{cases} \tag{4.14}$$

The coarse grid system-, respectively auxiliary-matrix can be computed by Galerkin's method:

$$\mathbf{K}_H = \mathbf{B}_H = (\mathbf{P}_h^{\text{sys}})^T \mathbf{K}_h \mathbf{P}_h^{\text{sys}}. \tag{4.15}$$

Smoothing

For the scalar case, a simple Gauss Seidel- or damped Jacobi-smoother can be applied, as defined in (2.3) and the algorithm is presented in Section 6.2.1.

Convergence Comparisons

The implemented Matlab algorithm for scalar $H^1(\Omega)$ problem is compared to a Matlab-intern PCG and GMRES solver. Fig.4.2.

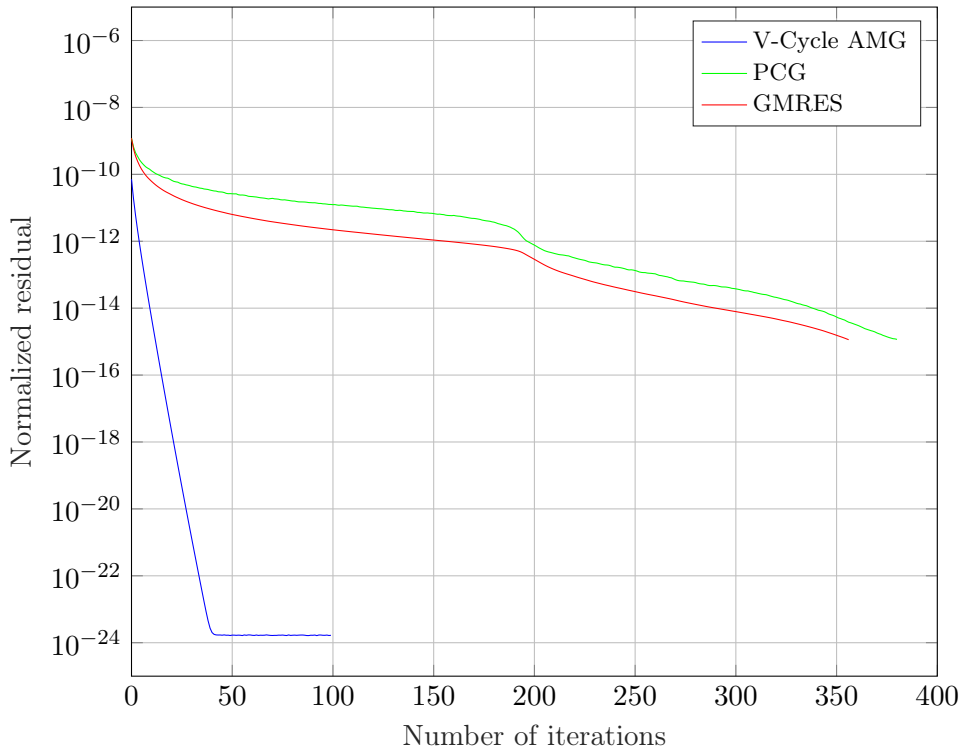

Figure 4.2: Convergence Comparison of scalar elliptic problem

4.3 $(H^1(\Omega))^p$ elliptic problems

This is the more general case of p-dimensional $H^1(\Omega)$-problems. An example is *Navier's equation* in the static case

$$\mathbf{f}_V + \nabla \cdot [\sigma] = 0. \tag{4.16}$$

Applying a linear constitutive equation $[\sigma] = [\mathbf{c}] : [\mathbf{S}]$, with $[\mathbf{c}]$ as the tensor of elasticity and $[\mathbf{S}]$ the strain-tensor, we obtain

$$\mathbf{f}_V + \nabla \cdot ([\mathbf{c}] : [\mathbf{S}]) = 0. \tag{4.17}$$

The weak formulation (in Voigt-notation and the differential operator matrix $\mathcal{B}$, as defined in eq. (5.12) of Section 5.2) follows as:

$$\int_\Omega (\mathcal{B}\mathbf{u}')^T [\mathbf{c}] \mathcal{B}\mathbf{u} d\Omega = \int_\Omega \mathbf{u}' \cdot \mathbf{f}_V d\Omega \tag{4.18}$$

with Dirichlet boundary conditions at the outer boundary. In this equation (4.18) we see the vector valued displacement-unknowns $\mathbf{u}$ and the test function $\mathbf{u}'$. For p-dimensional vector-unknowns, also the entries of the system matrix are vector valued: $(\mathbf{K}_h)_{ij} = k_{ij} \in \mathbb{R}^{p\times p}$. This means the size of the system matrix (number of unknowns) is $N_h \cdot p$, with N_h as the number of nodes. For a more detailed description of the physical field, see Chapter 5.

Auxiliary Matrix B_h

Again there are two ways to do an AMG-setup for the auxiliary matrix, the classic- or the geometric-way. The classic approach would be to define an appropriate matrix norm (4.19) and take the value of the norm as the entry of the system matrix

$$(\mathbf{B}_h)_{ij} = -||\mathbf{k}_{ij}||_\infty \quad i \neq j, \tag{4.19}$$

where $||\cdot||_\infty$ is the infinity-norm, defined as the maximum row-sum $||\mathbf{K}_h||_\infty = \max_j \sum_{i=1}^{N_h} |a_{ij}|$. Note that $\mathbf{B}_h \in \mathbb{R}^{N_h \times N_h}$, while $\mathbf{K}_h \in \mathbb{R}^{N_h \cdot p \times N_h \cdot p}$ and the norm includes the entries of the vector-valued DOF's at the specific node. For example, node i in Fig. 4.1 (assume it is a 3D-problem) has three unknowns, in the above weak form they are displacements and the corresponding entries in the system matrix form a $\mathbb{R}^{3\times 3}$ submatrix, e.g.

$$\tilde{\mathbf{k}}_{ij} = \mathbf{K}_h(\mathbf{i},\mathbf{j}) \begin{pmatrix} 3 & 1 & 0 \\ 1 & 4 & 9 \\ 0 & 9 & 2 \end{pmatrix}, \tag{4.20}$$

where $\mathbf{i}$ and $\mathbf{j}$ are the submatrix entries of the system matrix and correspond to the entries i, j in the auxiliary matrix the following way:

$$\begin{aligned} \mathbf{i} &= [3i+0, 3i+1, 3i+2] \\ \mathbf{j} &= [3j+0, 3j+1, 3j+2]\,. \end{aligned}$$

Finally the entry b_{ij} in the auxiliary matrix is obtained, using eq. (4.19)

$$b_{ij} = -||\tilde{\mathbf{k}}_{ij}||_\infty = -14. \tag{4.21}$$

The problem with this method is that it is less stable and needs more iterations to converge, especially for linear elasticity problems.

The second way is to define the auxiliary matrix based on geometry information, similar to (4.13). And again, as for the scalar case, the patterns of $\mathbf{K}_h$ are equal to the pattern of $\mathbf{B}_h$, since every off-diagonal $(p \times p)$-submatrix in $\mathbf{K}_h$ must represent the connection of two nodes. If we have a look at Fig. 4.3, we clearly see a connection between e.g. nodes ω_1 and ω_5, therefore the submatrix $\tilde{\mathbf{k}}_{1,1}$ of $\mathbf{K}_h$ has a off-diagonal sub-matrix $\tilde{\mathbf{k}}_{1,5}$. Also the entry in the auxiliary-matrix $b_{1,5}$ is non-zero and has the value of the distance between these nodes. On the other hand node ω_1 has no direct connection to node ω_9 and therefore the off-diagonal submatrix $\tilde{\mathbf{k}}_{1,9}$ is a 3×3 zero-matrix. This shows, in a heuristical way, that the auxiliary matrix inherits the graph of the underlying original system. An important notice is that the above considerations only hold if the nodes of the physical/spatial discretization (mesh) coincides with the discretization of the FE-space, which is true for first order Lagrangian finite elements. If higher order Lagrangian ansatz-functions are used, there are also nodes (at the zeros of the Lagrangian-polynomial) in the FE-space, which do not coincide with physical grid points. Therefore connections between nodes can no longer be interpreted as edges of a mesh. Nevertheless the information of the position of nodes in a higher-order FE-framework is known[5] and the above procedure should work because the graph of the system matrix actually represents a virtual mesh, which can be coarsened but this neighter proven in this work, nor further elaborated on.

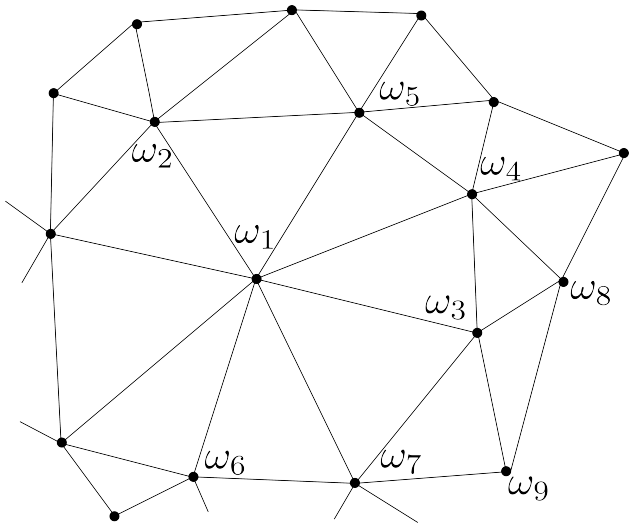

Figure 4.3: Node-connections in a FE-mesh

Prolongation

The difference to the scalar case is that the prolongation operators for the system matrix $\mathbf{P}_h^{\text{sys}}$ and the auxiliary matrix $\mathbf{P}_h^B$ are not the same anymore. The difference is that every entry of

[5] at least for nodal finite elements, the positions are located at discrete points

the auxiliary matrix corresponds to a $p \times p$-submatrix in the system-matrix, despite this fact, the operators are similar to (4.14), which is a simple piecewise constant interpolation. Actually the operator for $\mathbf{B}_h$ is exactly the same. For the system-matrix it reads as

$$(\mathbf{P}_h^{sys})_{ij} = \begin{cases} \mathbf{I}_p, & \text{if} \quad i = j \in \omega_C^n \\ \frac{1}{|S_h^i \cap \omega_C^n|} \cdot \mathbf{I}_p, & \text{if} \quad i \in \omega_F^n, j \in S_h^{i,T} \cap \omega_C^n \\ 0, & \text{else} \end{cases} \tag{4.22}$$

with $\mathbf{I}_p \in \mathbb{R}^{p \times p}$ as the p-dimensional identity matrix. Other, more sophisticated but also computationally more expensive prolongation operators are given in [6] and [9].
Analog to the scalar case, the coarse auxiliary- and system-matrices $\mathbf{K}_H$ and $\mathbf{B}_H$ are computed by Galerkin's method (4.15).

Smoothing

Again, we can use the same kind of smoother as in the scalar-case, which is the Gauss-Seidel or damped-Jacobi method. Nevertheless a *block*-version must be used, otherwise we obtain poor convergence.
This block-version can easily be constructed, by re-arranging the equations, such that for each node/point the spatial entries are inserted one after another. For example, the solution (displacement) vector with three nodes i, before the re-arrangement, might look like $\underline{u}_h = [\underline{u}_i, \underline{v}_i, \underline{w}_i] = [u_1, u_2, u_3, v_1, v_2, v_3, w_1, w_2, w_3]$, which would result in poor convergence. The correct ordering would be $\underline{u}_h = [u_1, v_1, w_1, u_2, v_2, w_2, u_3, v_3, w_3]$.

4.3.1 Convergence Comparisons

The implemented Matlab algorithm for a scalar $H^1(\Omega)$ problem is compared to a Matlab-intern PCG and GMRES solver, see Fig.4.2. For this convergence graphs, the weak form of system (4.17) is solved for the displacements.

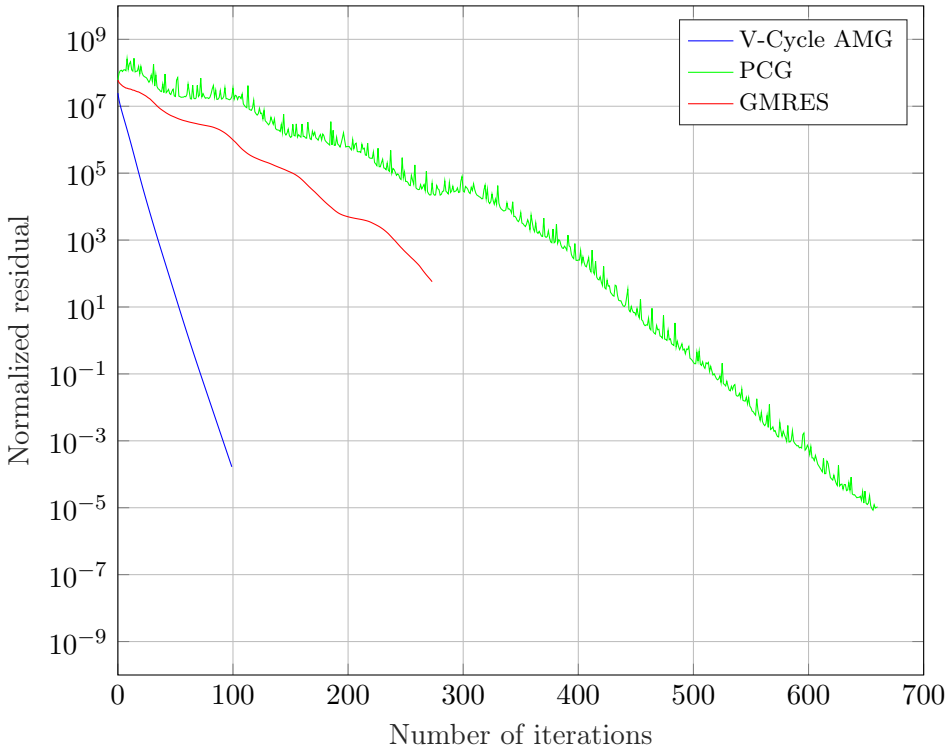

Figure 4.4: Convergence Comparison for static Navier's equation

4.4 $H(\mathbf{curl}, \Omega)$ elliptic problems

Since this function space is mainly used for edge-elements in computational electromagnetics, we solve the system of Maxwell's equations for the quasistatic case (often called eddy current case), where the displacement current density term is neglected. The strong form, where also moving bodies are neglected, reads as

$$\gamma \frac{\partial \mathbf{A}}{\partial t} + \nabla \times \nu \nabla \times \mathbf{A} = \mathbf{J}_i. \tag{4.23}$$

The weak form with appropriate boundary conditions (not mentioned here, because only the convergence for a generic example shall be shown) follows

$$\int_\Omega \gamma \mathbf{A}' \cdot \frac{\partial \mathbf{A}}{\partial t} d\Omega + \int_\Omega \nabla \times \mathbf{A}' \cdot \nu \nabla \times \mathbf{A} d\Omega = \int_\Omega \mathbf{A}' \cdot \mathbf{J}_i d\Omega. \tag{4.24}$$

For a more detailed description of the physical field, see Chapter 5.

Auxiliary Matrix B_h

In [7] it is mentioned that the coarsening can be performed on the nodes, like for the classical nodal Lagrange elements. Therefore we again introduce an auxiliary matrix $\mathbf{B}_h$, with the same "meaning" as in the two cases before but with a slightly different construction. For the $H^1(\Omega)$- and $(H^1(\Omega))^p$-version, the entries of $\mathbf{B}_h$ can be computed by taking the appropriate entry or block in the system matrix and perform a certain operation on it, including the geometry information. For the $H(\mathrm{curl}, \Omega)$-space, an element-wise assembly of the different contributions (4.25) has to

be carried out, where ν_r is the reluctivity of the material.

$$b_{ij}^r = -\frac{\nu_r}{||\mathbf{a}_{ij}||_2} \quad \text{with} \quad i \neq j \quad \text{and} \quad (i,j) \in \omega_h^e \tag{4.25}$$

The correct construction of this auxiliary matrix is explained in detail in Section 6.1.1.
Furthermore, we have to preserve the kernel of the curl-operator on the different virtual meshes, which is later on assured by constructing a special prolongation operator for the system matrix.

Prolongation

For the different prolongation operators, we have to introduce an additional construct, called the *index-map* ind(), which is defined by

$$\Omega_H^n = \text{ind}(\Omega_C^n). \tag{4.26}$$

Together with another coarsening-method, the agglomeration-method, described in Section 3.2.2, we can define the coarse nodes as

$$\Omega_h^n = \{\text{ind}(i) \quad | \quad i \in \Omega_h^n\}, \tag{4.27}$$

which means that all nodes in Ω_h^n are assigned an index, of the according agglomerate

$$I_h^i = \{j \in \Omega_h^n \quad | \quad \text{ind}(j) = \text{ind}(i)\} \subset N_h^i.$$

For example in Fig. 3.3, all nodes in agglomerate I_h^i are assigned the same index.
Since we use direct interpolation, the prolongation operators consist solely of 0 and 1 entries. The prolongation operator for the auxiliary matrix can be defined as

$$(P_h^B)_{ij} = p_{ij}^n = \begin{cases} 1 & \text{if} \quad i \in \Omega_h^n, j = \text{ind}(i) \\ 0 & \text{otherwise} \end{cases} \tag{4.28}$$

The coarse grid auxiliary matrix $\mathbf{B}_H$ is calculated via Galerkin's method, as for the two versions before.
Now we can deal with the construction of prolongation operators for the system matrix $\mathbf{K}_h$. It should be mentioned that every entry of the system matrix corresponds to a scalar quantity, defined on an edge of the finite element. The coarsening for the auxiliary matrix was carried out on nodes but now we face the problem that the system matrix is defined on edges, which introduces a more complex prolongation operator because not only the decrease of the number of edges should be provided but also the kernel of the curl-operator has to be.
Reitzinger [6] states that there is no proof yet, which manifests that the prolongation operators effectively decrease the number of edges in the coarse mesh. If however the number of non-zero entries in the auxiliary matrix does not grow too fast, the decrease of edges is heuristically given.
The second requirement, the preservation of the kernel, is ensured by constructing the prolongation operator for the system matrix in the following way (by assuming a positive orientation of

an edge $j = (j_1, j_2)$ from j_1 to j_2 if $j_1 < j_2$ holds):

$$(P_h^{sys})_{ij} = \begin{cases} 1, & \text{if} \quad j = (\text{ind}(i_1), \text{ind}(i_2)) \\ -1, & \text{if} \quad j = (\text{ind}(i_2), \text{ind}(i_1)) \end{cases} \tag{4.29}$$

Smoothing

Point- or simple block-smoothers, which were used for scalar or vectorial H^1-problems are not suitable anymore. Instead of them, an overlapping technique must be used, which smoothes all edges, connected to one node, together, see Fig.4.1. Similar to [9] we can use a connectivity matrix $\mathbf{R}_j$, which extracts the appropriate subblocks $\mathbf{K}^j$ out of the full system matrix $\mathbf{K}_h$ by applying

$$\mathbf{K}^j = \mathbf{R}_j \mathbf{K}_h \mathbf{R}_j^T. \tag{4.30}$$

Now, we can apply a Gauss-Seidel smoother to every node $j = 1, ..., n$ of the fine mesh

$$\underline{u}_{i+1}^j = \underline{u}_i^j + \mathbf{R}_j^T (\mathbf{K}^j)^{-1} \mathbf{R}_j (\underline{f}_h - \mathbf{K} \cdot \underline{u}_i). \tag{4.31}$$

Since the extraction of the subblocks and the sub-residuals can already be performed in the setup-phase, including the evaluation of the quadratic form, the performance is only slightly worse than a standard point Gauss-Seidel method. For details about implementation, see Section 6.2.2. The effects of the smoother in this case, should not be neglected, because choosing, e.g. a standard point-Gauss-Seidel method can destroy the good convergence of the solver, as shown later on in Fig. 5.8 in Section 5.3.3.

4.4.1 Convergence Comparisons

The implemented Matlab algorithm for a $H(\text{curl}, \Omega)$ problem is compared to a Matlab-intern PCG and GMRES solver and the results are displayed in Fig.4.2.

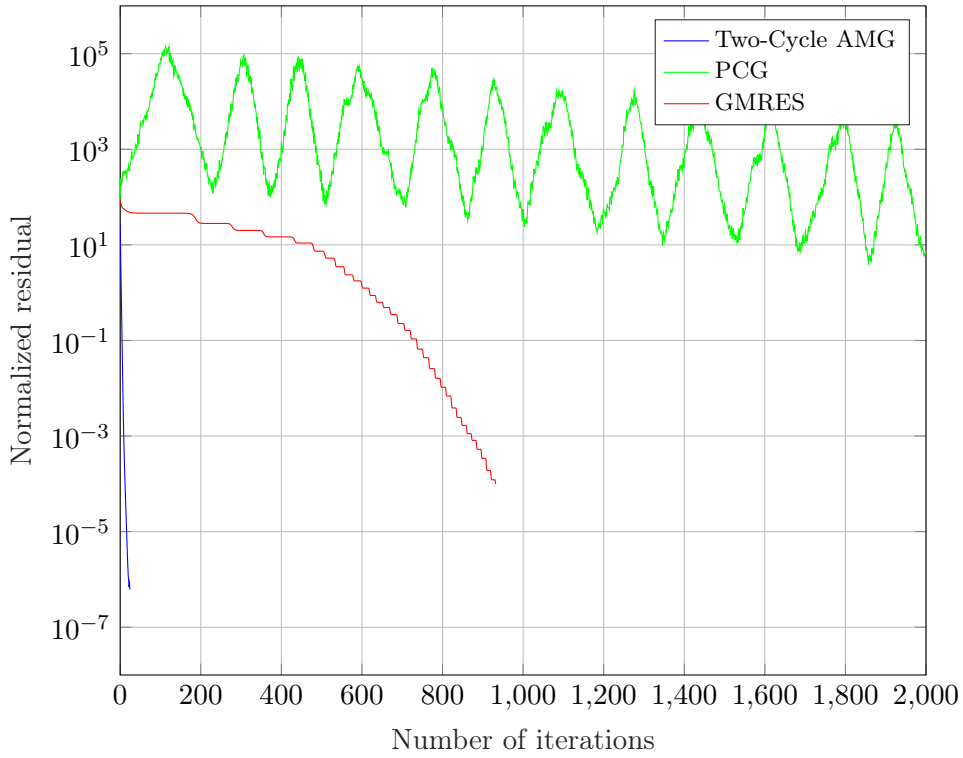

Figure 4.5: Convergence comparison for electromagnetic problem

Above convergence plots show promising results for the implemented AMG-versions and the next step is to implement these algorithms in the in-house FEM-code CFS++ and try to observe similar convergence properties and also investigate the execution time for solving the systems. In the next chapter, different application examples, in the three function spaces from above are used to test the CFS++ implementations, using AMG as a preconditioner for a CG-solver.

Chapter 5

Application Examples

All results in this chapter are computed, using a PC with an Intel i7-3820 3.6GHz processor and using only one OpenMP-thread, in order to make different solvers comparable, which might have better or worse parallelization. The compiler, used to build CFS++ was gcc-4.8.5, with optimization flag -O3.

5.1 Electric potential: Capacitor

In this example, a capacitor, depicted in 5.1, with two electrodes and prescribed scalar potential is solved, using an AMG-preconditioner for a CG-solver. The medium inside the capacitor has a relative permeability of 1. It should be mentioned that the following examples have no deeper physical meaning, they shall only be used to verify the AMG algorithms.

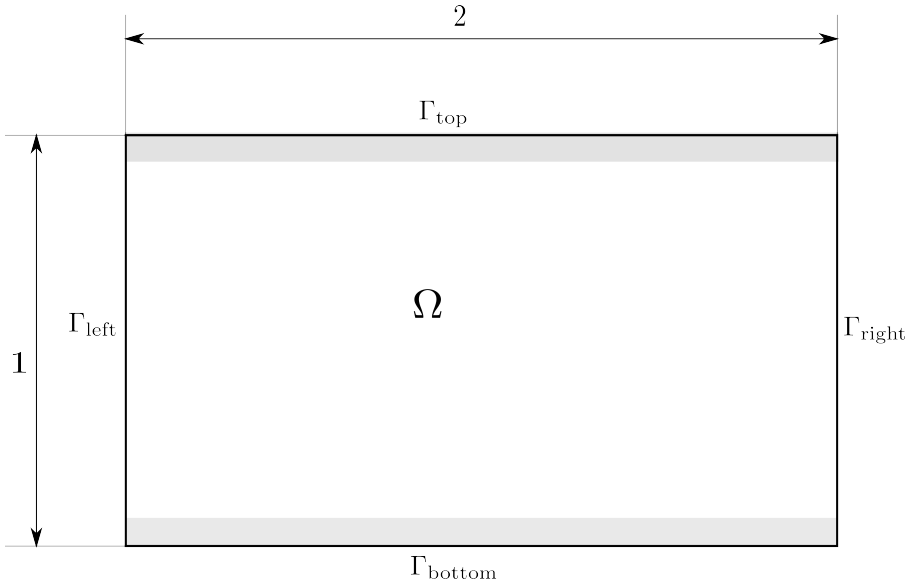

Figure 5.1: Capacitor setup

5.1.1 Physical Description

The equations, which need to be solved for this case are the *Maxwell equations* for the electrostatic case

$$\nabla \times \mathbf{E} = 0 \tag{5.1}$$

$$\nabla \cdot \mathbf{D} = q_\mathrm{e} \tag{5.2}$$

$$\mathbf{D} = \epsilon \mathbf{E}. \tag{5.3}$$

Because the electric field intensity $\mathbf{E}$ is curl-free, the Helmholtz-decomposition[1] can be applied and $\mathbf{E}$ is purely defined by the scalar potential V_e

$$\mathbf{E} = -\nabla V_\mathrm{e}. \tag{5.4}$$

By inserting this equation into (5.3), we obtain

$$-\nabla \cdot \epsilon \nabla V_\mathrm{e} = q_\mathrm{e}. \tag{5.5}$$

Now the problem in the strong form can be stated, similar to [9], as

Given :

$q_\mathrm{e} = 0$

$\epsilon : \Omega \to \mathbb{R}$

Find : $V_\mathrm{e} : \Omega \to \mathbb{R}$

$-\nabla \cdot \epsilon \nabla V_\mathrm{e} = q_\mathrm{e}$

Boundary Conditions :

$V_\mathrm{e} = 0V \quad \text{on} \quad \Gamma_\mathrm{bottom}$

$V_\mathrm{e} = 10V \quad \text{on} \quad \Gamma_\mathrm{top}$

5.1.2 Weak Formulation

Since we have prescribed Dirichlet boundary conditions, our solution V_e and test function v are elements of Sobolev-spaces, which fulfill the boundary conditions:

$$V_e \in W(\Omega) = \{w \in H^1(\Omega) \mid w = 0 \text{ on } \Gamma_\mathrm{bottom},\ w = 10 \text{ on } \Gamma_\mathrm{top}\}, \tag{5.6}$$

$$v \in H_0^1(\Omega). \tag{5.7}$$

[1] Let $\mathbf{u} \in V$, be a vector on a bounded domain, then vector $\mathbf{u}$ can be split up into a rotation-free and a divergence-free part

$$\mathbf{u} = -\nabla\Phi + \nabla \times \mathbf{A},$$

with Φ as a scalar potential and $\mathbf{A}$ as a vector potential.

By performing an integration by parts and use the property of the test function (it must vanish at Dirichlet boundaries), we obtain the weak formulation as

$$\int_{\Omega} \epsilon \nabla v \cdot \nabla V_e d\Omega = 0. \tag{5.8}$$

Then the Galerkin approach, with linear Lagrange ansatz-functions, is applied and results in the final linear system

$$\mathbf{K} \cdot \underline{V}_e = \underline{f}. \tag{5.9}$$

At this point it is important to notice that the whole AMG-framework in Matlab and CFS++ only works with lowest-order, linear ansatz-functions.

5.1.3 Computational Setup and Results

The computational domain is meshed with 40000 regularly structured linear quadrilateral elements. To explore the convergence of the method and compare it to a classical GMRES and CG solver, the residual in every iteration step is stored and depicted in Fig. 5.2. In this plot, we can clearly see the simplicity of the setup. The residuals of all three solver-types start at an already very low residual of about 10^{-9}, which is due to the two-dimensional setup and the overal good property of the Laplace operator in an FE-context. Furthermore we can observe that, although the initial residual is very low, the pure CG solver has obvious problems to decrease the residual, whereas GMRES and the AMG-preconditioned CG-solver have some potential left to minimize the residual. Even in this simple example, we can see a constant logarithmic decrease of the residual for the AMG-CG version, whereas the graph for GMRES becomes less steep, the lower the residual gets.

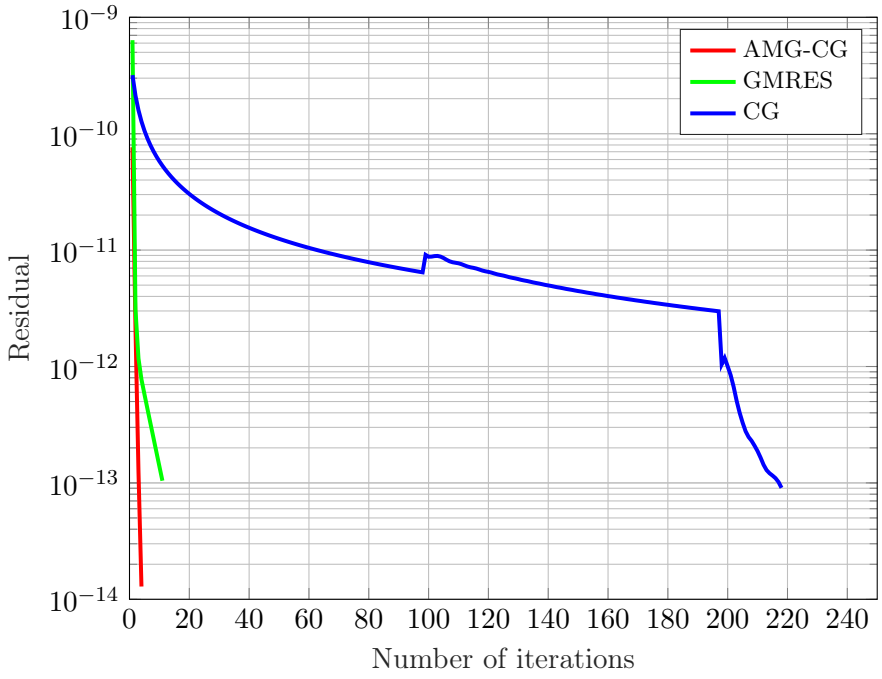

Figure 5.2: Convergence comparison for the 2D capacitor problem

The AMG-algorithm performed a splitting into five hierarchy-levels, consisting of 40000 nodes on the finest level, 10000 on level 2, 2500 on level 3, 607 on level 4 and 150 on the coarsest one. To show the benefits not only in the "residual-space" but also in terms of wallclock-times, the number of nodes was increased and the CPU-times are stored and compared to a GMRES-solver. The results are presented in Table 5.1, which shows the extremely good $\mathcal{O}(N)$ complexity of the AMG-algorithm, because the factor between size of system two and size of system one is 2.25 and the solve-times have a factor of 2.3. GMRES on the other hand cannot stay compatible for larger systems, since it's complexity is higher.
At this place it should be mentioned, that the setup-phase has to be computed only once for the whole analysis and can become quite large, especially for the vector-valued and edge-version. The solve-time is more important, since this step has to be performed e.g. in every time-step of a transient analysis.

N_h	GMRES		AMG-CG		
	Setup	Solve	Setup	Solve	Nr. of Levels
40000	0.64	3.96	0.62	0.23	5
90000	1.43	15.44	1.42	0.53	6

Table 5.1: Wallclock-times of different system-sizes

5.2 Mechanical Field: Loaded Beam

In this example, a cantilever beam, fixed at the left surface Γ_{left} and loaded with a constant pressure at the top-surface Γ_{top} is computed. As in the example before, an AMG-CG is used to solve the problem. The isotropic material parameter are chosen to be $E = 210000\text{MPa}$ for the E-modulus and $\nu = 0.3$ for the Poisson-number.

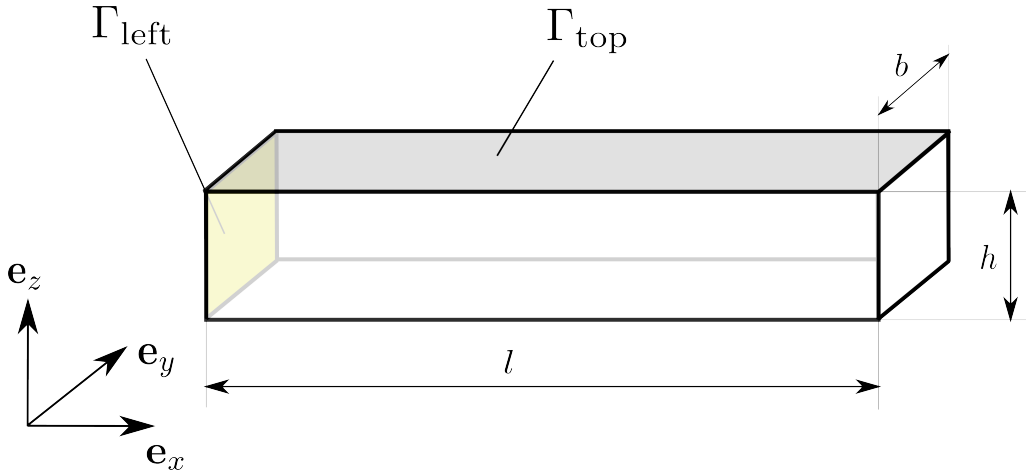

Figure 5.3: Loaded beam

5.2.1 Physical Description

The equations, which need to be solved for this case is *Navier's equation*

$$\mathbf{f}_V + \nabla \cdot [\sigma] = \rho \mathbf{a}, \tag{5.10}$$

which can be derived as presented in [9]. In this equation, $[\sigma]$ denotes the *Cauchy stress tensor*, which can be expressed, as already mentioned in Section 4.3, as the scalar-product[2] of the tensor of elasticity $[\mathbf{c}]$ and the strain-tensor $[\mathbf{S}]$. The volumetric forces are denoted by $\mathbf{f}_V$. In an FE-context, it is more convenient to represent the stress-tensor $[\sigma]$ as a vector σ of its tensor-components, called *Voigt-notation*

$$[\sigma] = \begin{bmatrix} \sigma_{xx} & \sigma_{xy} & \sigma_{xz} \\ \sigma_{yx} & \sigma_{yy} & \sigma_{yz} \\ \sigma_{zx} & \sigma_{zy} & \sigma_{zz} \end{bmatrix} \rightarrow \sigma = \begin{pmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{zz} \\ \sigma_{yz} \\ \sigma_{xz} \\ \sigma_{xy} \end{pmatrix}. \tag{5.11}$$

Now, we can transfer the divergence operator of eq. (5.10), acting on the tensor into the equivalent matrix-form and introduce the differential-operator $\mathcal{B}$ as in [9]

$$\mathcal{B} = \begin{pmatrix} \frac{\partial}{\partial x} & 0 & 0 & 0 & \frac{\partial}{\partial z} & \frac{\partial}{\partial y} \\ 0 & \frac{\partial}{\partial y} & 0 & \frac{\partial}{\partial z} & 0 & \frac{\partial}{\partial x} \\ 0 & 0 & \frac{\partial}{\partial z} & \frac{\partial}{\partial y} & \frac{\partial}{\partial x} & 0 \end{pmatrix}^T. \tag{5.12}$$

Eq. (5.10) can now be re-written to

$$\mathcal{B}^T \sigma + \mathbf{f}_V = \rho \mathbf{a}. \tag{5.13}$$

Since we want to solver for the mechanical displacements $\mathbf{u}$ and not for the stress-components, we have to introduce the definition of the linear mechanical strain tensor $\mathbf{S}$ in eq. (5.14), which is derived from the full *Green-Lagrange strain-tensor* linearization and neglecting higher order terms, for the full formulation, see [9].

$$[\mathbf{S}] = \frac{1}{2}\left(\nabla_X \mathbf{u} + (\nabla_X \mathbf{u})^T\right) = \begin{bmatrix} \frac{\partial u_x}{\partial x} & \frac{1}{2}\left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x}\right) & \frac{1}{2}\left(\frac{\partial u_x}{\partial z} + \frac{\partial u_z}{\partial x}\right) \\ \frac{1}{2}\left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x}\right) & \frac{\partial u_y}{\partial y} & \frac{1}{2}\left(\frac{\partial u_y}{\partial z} + \frac{\partial u_z}{\partial y}\right) \\ \frac{1}{2}\left(\frac{\partial u_x}{\partial z} + \frac{\partial u_z}{\partial x}\right) & \frac{1}{2}\left(\frac{\partial u_y}{\partial z} + \frac{\partial u_z}{\partial y}\right) & \frac{\partial u_z}{\partial z} \end{bmatrix} = s_{ij} \tag{5.14}$$

where ∇_X denotes the derivatives with respect to the original coordinates (Lagrangian coordinates). This tensor can also be written in *Voigt notation* $[\mathbf{S}] \rightarrow \mathbf{S}$ and by applying the differential

[2] Scalar product of two tensors $\mathbf{A}$ and $\mathbf{B}$ is defined the following way:

$$\mathbf{A} : \mathbf{B} = A_{ij} B_{ij}$$

operator $\mathcal{B}$ on $\mathbf{u}$, we obtain the strain tensor $\mathbf{S}$ in Voigt-notation.
By putting all definitions from above together and inserting it into the original Navier's equation (5.10), we obtain the Navier's-equation in (computational-friendly) Voigt-notation:

$$\mathbf{f}_V + \mathcal{B}^T[\mathbf{c}]\mathcal{B}\mathbf{u} = \rho\frac{\partial^2\mathbf{u}}{\partial t^2}. \tag{5.15}$$

Now the problem in the strong form can be stated as (static case)

Given :

$h = b = 1\text{m}, \quad L = 5\text{m}$

$\mathbf{a} = \mathbf{0}$ static case

$\rho =$ undefined, since $\mathbf{a} = \mathbf{0}$

$E = 210000\text{MPa}$ (isotropic material)

$\nu = 0.3$

Find : $\mathbf{u} : \Omega \to \mathbb{R}^3$

$\mathbf{f}_V + \mathcal{B}^T[\mathbf{c}]\mathcal{B}\mathbf{u} = \mathbf{0}$

Boundary Conditions :

$\mathbf{u} = \mathbf{0}$ on Γ_{left}

$[\sigma]^T \cdot \mathbf{n} = -20000\frac{\text{N}}{\text{m}^2}\mathbf{e}_z$ on Γ_{top}

5.2.2 Weak Formulation

Due to the boundary conditions at the boundary of the computational domain Ω, the solution quantity $\mathbf{u} \in (H_0^1)^3$ and test function $\mathbf{v} \in (H_0^1)^3$ are elements of an appropriate Sobolev-space, which incorporates the boundary condition, as in the scalar case. By performing an integration by parts and use some properties, imposed on the test function, as in the electrostatic-case, we otain the weak formulation as

$$\int_\Omega (\mathcal{B}\mathbf{v})^T[\mathbf{c}]\mathcal{B}\mathbf{u}d\Omega = \int_\Omega \mathbf{v}\cdot\mathbf{f}_V d\Omega. \tag{5.16}$$

The non-trivial kernel[3] of the underlying bilinear form of eq. (5.16)

$$a(\mathbf{u},\mathbf{v}) = \int_\Omega (\mathcal{B}\mathbf{v})^T[\mathbf{c}]\mathcal{B}\mathbf{u}d\Omega \tag{5.17}$$

forms, according to [6], a six-dimensional subspace (rigid body modes)

$$\ker(a(\mathbf{u},\mathbf{v})) = \{\mathbf{u}\in\mathbb{V} \mid \mathcal{B}\mathbf{u} = 0\ \forall\mathbf{v}\in\mathbb{V}\} = \{\mathbf{a}\times\mathbf{x}+\mathbf{b} \mid \mathbf{a},\mathbf{b}\in\mathbb{R}^d\}. \tag{5.18}$$

[3] A kernel is called *trivial* if it consists only of the nullvector of the vectorspace

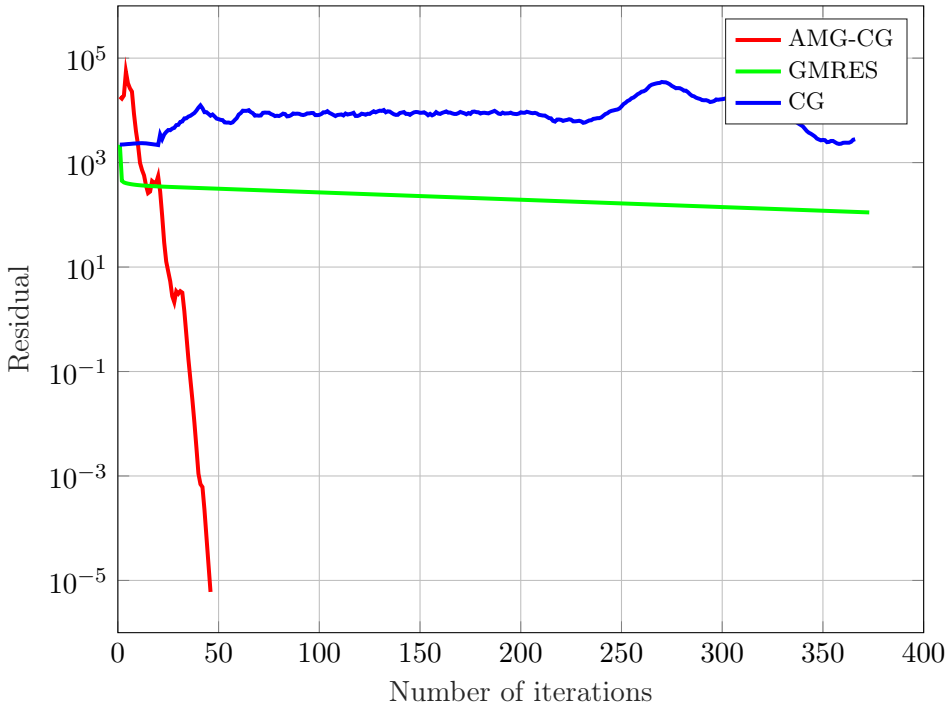

Figure 5.4: Convergence comparison between AMG-CG, GMRES and CG

Applying the Galerkin approach with linear Lagrange Ansatz-functions, we obtain the final linear system

$$\mathbf{K}_u \cdot \mathbf{u} = \mathbf{f}, \tag{5.19}$$

where $\mathbf{K}_u$ is the system matrix, which consists only of the stiffness-matrix, since we have no acceleration (no mass-matrix) and no damping (no damping-matrix).

5.2.3 Computational Setup and Results

The first analysis is performed, using 132300 undestorted hexahedral elements and the convergence between CG, GMRES and AMG-CG is compared. For the AMG-CG we are solving the problem on three hierarchy-levels. As one can see in Figure 5.4, the AMG-CG converges well, while GMRES and CG have serious problems to lower the residual, CG does not even seem to converge at all. The problem might be the strong connection between the different spatial-directions (x, y, z). Even the AMG-CG runs into difficulties, when the mesh gets more distorted. For example in Figure 5.5, the hexahedral element are distorted by a certain factor $\frac{\text{length in x-direction}}{\text{length in y-, z-direction}}$, as shown in the legend. As [6] states, equations arising from eq. (5.16) are challenging for AMG because of the nontrivial kernel (5.18) of the corresponding operator respectively bilinear form, which can not be approximated well enough. Furthermore, the prolongation-type we are using (averaged) is not the optimal one, because it only preserves constant functions, whereas the prolongation, based on harmonic extension is able to approximate more than constant functions [6].
Restricting the analysis to the undestorted case, we can investigate the scalability and check if $\mathcal{O}(N)$ is still reached. For this field, the results of AMG-CG are not compared to GMRES

because it would take too long to converge to the residual-threshold of 10^{-8}, which can be seen in Fig. 5.4. Looking at Table 5.2, it becomes obvious that the optimal complexity is no longer valid because due to the convergence-issues the number of iterations increases and therefore the solution-time rises over-proportional.

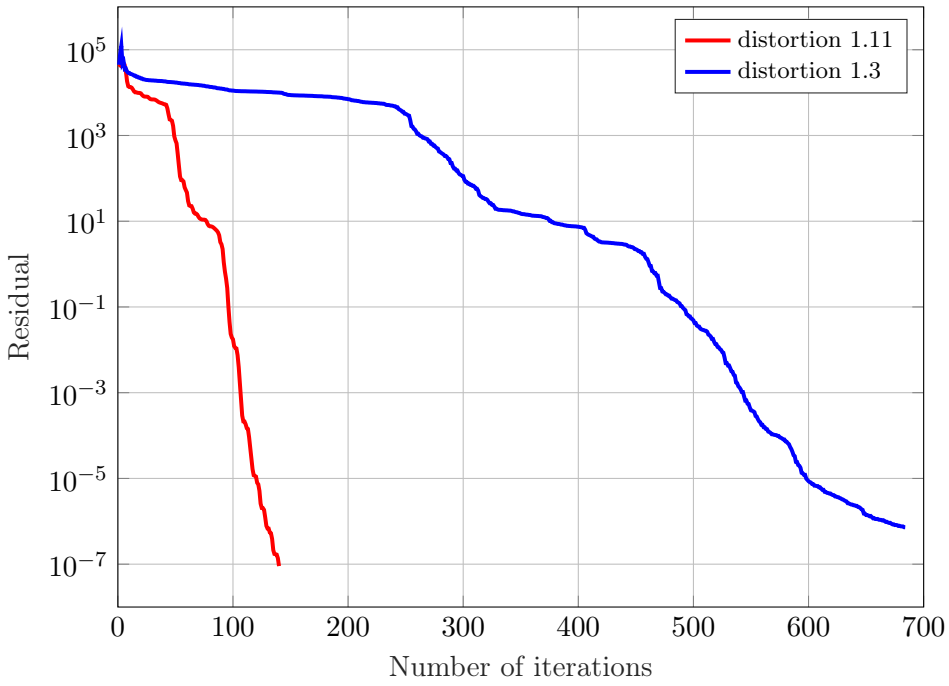

Figure 5.5: Convergence comparison using AMG-CG for different mesh distortions

N_h	AMG-CG		
	Setup	Solve	Nr. of Levels
132300	25.5	19.43	4
432450	83.95	216.95	5

Table 5.2: Wallclock-times for different system-sizes

5.3 Electromagnetic Field: Current Loaded Coil

In this example, the electromagnetic field of a current loaded copper coil with an iron-core and surround air, as depicted in Fig. 5.6, is computed. The difference to the two examples before is the kind of FE-space. Here, we are using Nédélec's edge elements, which are the natural choice of the $H(\mathrm{curl}, \Omega)$ space.

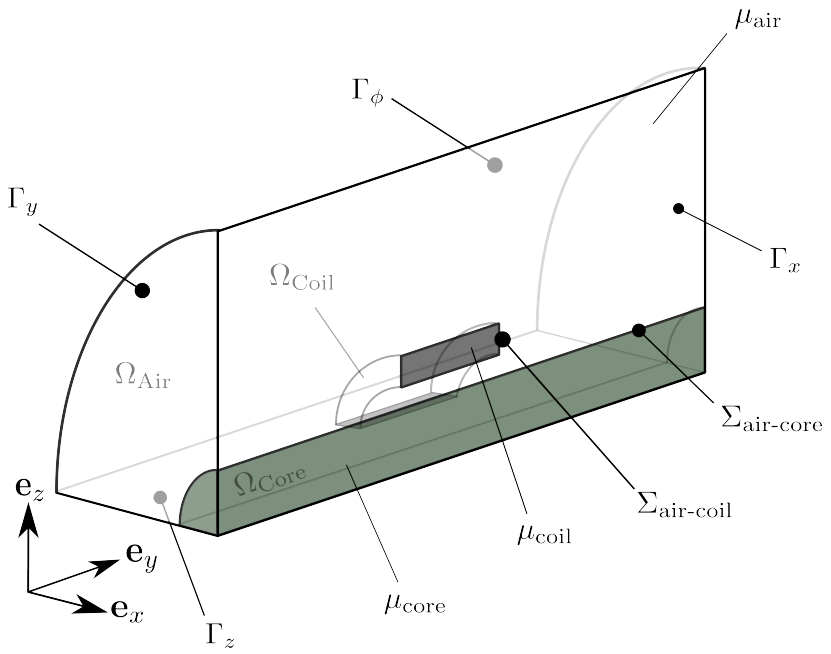

Figure 5.6: Current loaded coil

5.3.1 Physical Description

The set of equations, which needs to be solved for this example, are *Maxwell equations*

$$\nabla \times \mathbf{H} = \mathbf{J} + \frac{\partial \mathbf{D}}{\partial t}, \tag{5.20}$$

$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t}, \tag{5.21}$$

$$\nabla \cdot \mathbf{D} = q_e, \tag{5.22}$$

$$\nabla \cdot \mathbf{B} = 0, \tag{5.23}$$

together with the three constitutive equations

$$\mathbf{J} = \gamma(\mathbf{E} + \mathbf{v} \times \mathbf{B}), \tag{5.24}$$

$$\mathbf{D} = \epsilon \mathbf{E}, \tag{5.25}$$

$$\mathbf{B} = \mu \mathbf{H}. \tag{5.26}$$

Furthermore, we assume, that none of the three parts (coil, core, air) are moving ($\mathbf{v} = \mathbf{0}$) and all quantities are independent of time (static problem). In this case, all time derivatives in the Maxwell equations vanish. In this case, the electromagnetic field, as goverened by Maxwell equations splits up into an electrostatic- and a magnetic-part and the latter, which describes the physics of our problem, will be computed. In the next step, we introduce the magnetic vector potential $\mathbf{A}$ via

$$\mathbf{B} = \nabla \times \mathbf{A}. \tag{5.27}$$

We are allowed to do this, since $\mathbf{B}$ is solenoidal ($\nabla \cdot \mathbf{B} = 0$). Therefore, eq. (5.23) automatically fulfilled and inserting eq. (5.27) into eq. (5.26) and (5.20), we arrive at a curl-curl problem (static case)

$$\nabla \times \frac{1}{\mu} \nabla \times \mathbf{A} = \mathbf{J}, \tag{5.28}$$

with the reluctivity $\nu = \frac{1}{\mu}$.
Now the complete problem in the strong form can be stated as (static case):

Given :

$$\mu_{\text{air}} = 1.2566 \cdot 10^{-6} \frac{\text{Vs}}{\text{Am}}, \quad \mu_{\text{coil}} = 1.2566 \cdot 10^{-6} \frac{\text{Vs}}{\text{Am}} \quad \mu_{\text{core}} = 6.2 \cdot 10^{-3} \frac{\text{Vs}}{\text{Am}}$$

Find : $\mathbf{A} : \Omega \to \mathbb{R}^3$

$$\nabla \times \nu \nabla \times \mathbf{A} = \mathbf{J}_i$$

Boundary Conditions :

$$\mathbf{n} \times \mathbf{A} = 0, \quad \text{on} \quad \Gamma_\phi$$

$$\mathbf{n} \cdot \mathbf{A} = 0, \quad \text{on} \quad \Gamma_x, \Gamma_y, \Gamma_z$$

Interface Conditions at material parameter-discontinuities :

$$\mathbf{n} \cdot \mathbf{A}_{\text{coil}} = \mathbf{n} \cdot \mathbf{A}_{\text{air}}, \quad \text{on} \quad \Sigma_{\text{air-coil}}$$

$$\mu_{\text{air}}\, \mathbf{n} \times \nabla \times \mathbf{A}_{\text{air}} = \mu_{\text{coil}}\, \mathbf{n} \times \nabla \times \mathbf{A}_{\text{coil}}, \quad \text{on} \quad \Sigma_{\text{air-coil}}$$

$$\mathbf{n} \cdot \mathbf{A}_{\text{core}} = \mathbf{n} \cdot \mathbf{A}_{\text{air}}, \quad \text{on} \quad \Sigma_{\text{air-core}}$$

$$\mu_{\text{air}}\, \mathbf{n} \times \nabla \times \mathbf{A}_{\text{air}} = \mu_{\text{core}}\, \mathbf{n} \times \nabla \times \mathbf{A}_{\text{core}}, \quad \text{on} \quad \Sigma_{\text{air-core}}$$

5.3.2 Weak Formulation

Before we multiply with a test function $\mathbf{A}'$, an appropriate space has to be constructed. The classical $H_0(\text{curl}, \Omega)$ is not enough, since it does not include interface-restrictions. Therefore, we introduce an appended Sobolev-space

$$H_0^\Sigma = \{\mathbf{u} \in H_0(\text{curl}, \Omega) | \mathbf{u} \times \mathbf{n}|_\Sigma = 0\}. \tag{5.29}$$

After the appropriate function space is constructed and $\mathbf{A}, \mathbf{A}' \in H_0^\Sigma$ is stated, we can write the weak formulation as

$$\int_\Omega \nabla \times \mathbf{A}' \cdot \nu \nabla \times \mathbf{A} d\Omega = \int_\Omega \mathbf{A}' \cdot \mathbf{J}_i d\Omega. \tag{5.30}$$

As stated in [6], we have to add a term, scaled with a so-called fictitious condictivity $\gamma' << \nu$ in order to ensure the uniqueness of the boundary value problem. The additional term does not much effect the solution and it results in the regularized weak form

$$\int_\Omega \nabla \times \mathbf{A}' \cdot \nu \nabla \times \mathbf{A} d\Omega + \int_\Omega \gamma' \mathbf{A}' \cdot \mathbf{A} d\Omega = \int_\Omega \mathbf{A}' \cdot \mathbf{J}_i d\Omega. \tag{5.31}$$

The next step is the discretization of the geometry, followed by the discretization of the continuous

function-space into a finite space, which is done via so-called (*Nédélec*) *edge elements*. The main difference to Lagrangian ansatz-functions, without going into to much detail, is that degrees of freedom do not correspond with *nodes* but with *edges*. The classical Galerkin-ansatz, e.g. for the vector potential is written as

$$\mathbf{A} \approx \sum_{i=0}^{N_{edges}} \mathbf{N}_i \tilde{A}_i, \tag{5.32}$$

with N_{edges} as the number of edges in the domain, with no boundary condition (unknown vector potential), $\mathbf{N}_i$ as the edge- (Nédélec-) shape function corresponding to edge i and $\tilde{A}_i$ as the actual degree of freedom. This DOF shall not be confused with the component of $\mathbf{A}$ because it is the line-integral of the vector potential along edge i

$$\tilde{A}_i = \int_i \mathbf{A} \cdot d\mathbf{s}. \tag{5.33}$$

For a more rigorous introduction into edge elements, see [9] or [16]. Since the problem is static, there is no time derivative in the final formulation and together with the Galerkin-ansatz for edge-elements, the linear system can be derived as

$$\mathbf{K}_A \underline{A} = \underline{f}. \tag{5.34}$$

5.3.3 Computational Setup and Results

The first analysis for the convergence comparison between AMG-CG, GMRES and CG were carried out on a discretization with about 20000 edges. Due to the finite element formulation of edge elements, the size of the linear system was 17067, because on surface Γ_Φ, the magnetic flux density was forced to be tangential to the surface and therefore the vector-potential is zero. The convergence-results are depicted in Fig. 5.7, where the superiority of AMG-CG is clearly visible. The classical CG-method does not converge at all, as does the GMRES, or at least very slowly. To emphesize the importance for an appropriate smoother is shown if Fig. 5.8, where a standard point Gauss-Seidl method was used to solve the problem, instead of the correct Arnold-Falk-Winther (AFW) smoother. Also the computation times are some order of magnitudes lower than for GMRES, which is not even mentioned in Table 5.3 because it took too long to decrease the residual to a level, where it is comparable to AMG-CG. One abnormality was observed for the solution time of the largest system in Table 5.3, which is lower than the solution time for the next smaller system. It is not obvious where this comes from but it is definitely an artefact from implementation and not from the algorithmic-side. A possible explanation might be that the small systems did not fully utilize Intel-MKL's abilities in terms of internal storage schemes or optimizations regarding cache efficiency.

N_h	AMG-CG		
	Setup	Solve	Nr. of Levels
17067	3.04	1.4	2
23100	3.12	2.02	2
60304	19.37	16.2	2
114080	64.72	13.61	3

Table 5.3: Wallclock-times for different system-sizes

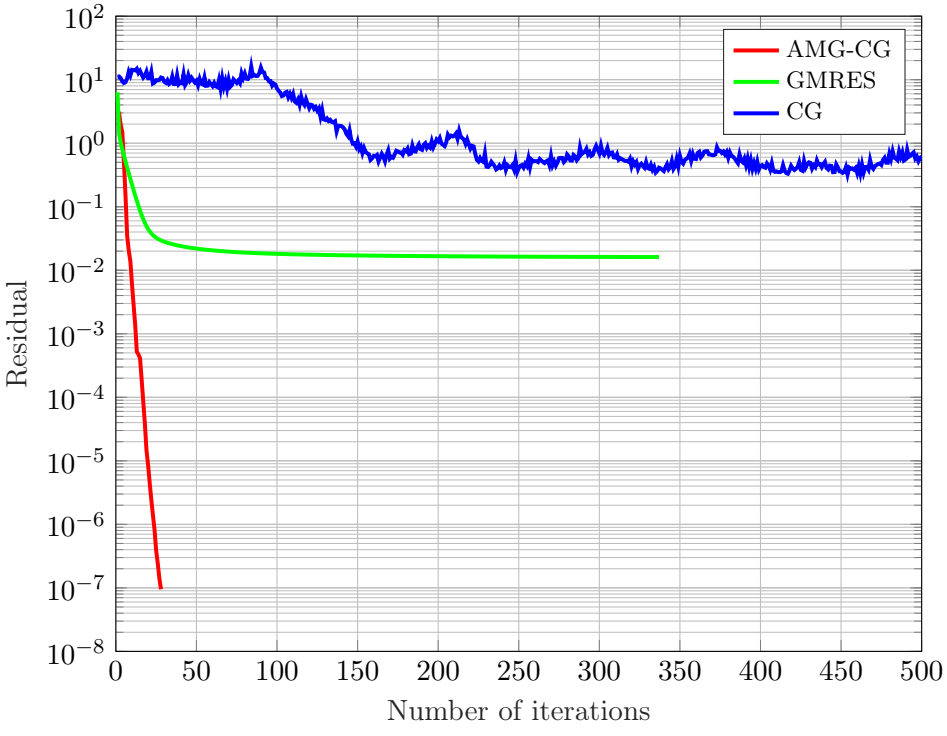

Figure 5.7: Convergence comparison between AMG-CG, GMRES and CG for the current loaded coil

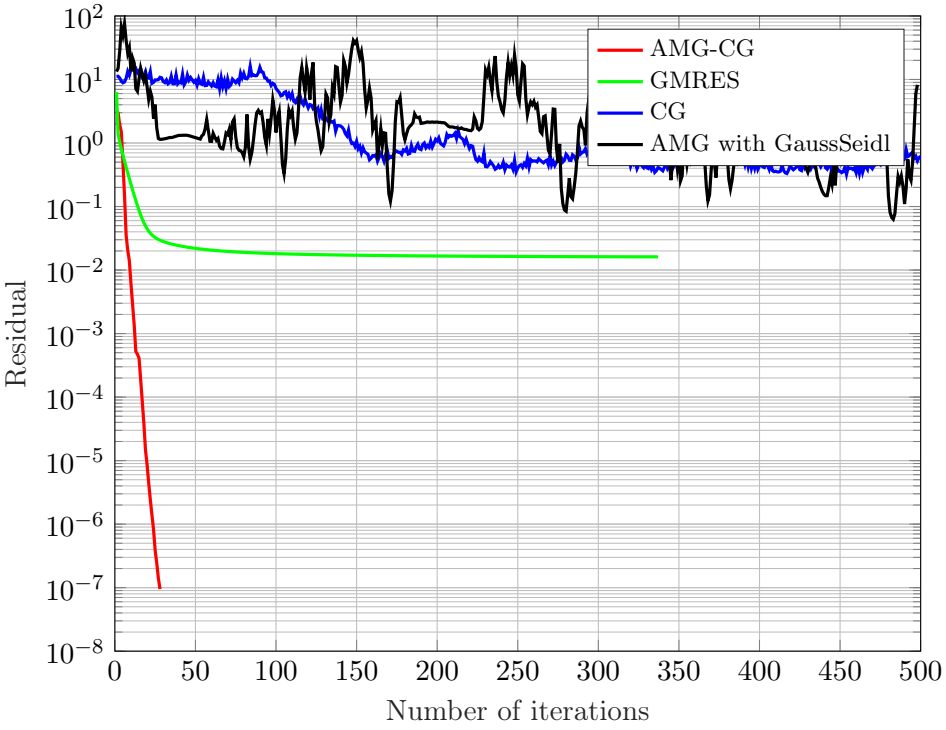

Figure 5.8: Convergence comparison with a wrong smoother

Chapter 6

Implementation Details

In this chapter, a short introduction into the implementation in CFS++ is given, with the main focus on the algorithmic approach and less on the sofware-development process, since there is definitely some room for improvement regarding this topic. The main goal of this chapter is to provide some basic informations to improve or extend the existing implementation, especially for mechanical engineers, who might be unfamiliar with some of the following terms.

6.1 Sparse Matrix Storage

Since we are nearly solely dealing with sparse matrices, the only feasible matrix storage format is a CRS- (compressed row storge) or CCS- (compressed column storage) storage format. These formats have the ability to store nnz non-zero entries of a $N \times N$-matrix in only $2 \cdot nnz + N + 1$ storage-entries instead of the N^2 storage-entries, when using a dense matrix storage. This format simply avoids, storing zero-entries in the matrix. Depending on the matrix type, the number of storage entries in the sparse-format can be furhter decreased, if the matrix is symmetric, then the format are called S(ymmetric)CRS or S(ymmetric)CCS.
The implementations in CFS++ were solely carried out with the non-symmetric CRS storage format. Although there are numerous explanations for this format, a short description is provided in the following.
CRS consists of three arrays: row-index rI, column-index cI and data-index dI. Assume we have, for example the following $(m \times n) = (3 \times 4)$ matrix

$$\mathbf{K} = \begin{pmatrix} 1 & 0 & 0 & 5 \\ 2 & 5 & 0 & 0 \\ 0 & 0 & 7 & 3 \end{pmatrix},$$

which has three rows and six non-zero entries. The row-index rI has size $(m+1)$ and stores for each row the number of non-zero elements, with its first entry 0, per definition. Following this explanation, we can construct the row-index as

$$rI = \begin{pmatrix} 0 & 2 & 4 & 6 \end{pmatrix}.$$

The column-index cI contains for each non-zero element the corresponding column in the matrix, in our example it would be (0-based indexing)

$$cI = \begin{pmatrix} 0 & 3 & 0 & 1 & 2 & 3 \end{pmatrix},$$

as does the data-index

$$dI = \begin{pmatrix} 1 & 5 & 2 & 5 & 7 & 3 \end{pmatrix}.$$

These definitions make it easy to loop over every non-zero element in the matrix (row-wise), according to pseudocode in Algorithm 6.

Algorithm 6 Looping over every non-zero element in a CRS-matrix

```
1: function CRS-LOOP
2:     for i = 0 : m do
3:         for j = rI[i] : rI[i + 1] do
4:             K(i, cI[j]) = dI[j]
5:         end for
6:     end for
7: end function
```

An important fact, which is especially beneficial for iterative solver, like Gauss-Seidl or Jacobi is the knowledge where the diagonal elements of a quadratic ($N \times N$) matrix are. There are two ways (implemented in CFS++) for this purpose. The first method introduces a new array $diagI$, which stores for each diagonal entry the position in the data array. The second method performes a re-ordering of cI and dI, so that the diagonal element of row i is stored at position $rI[i]$. The second version makes it very easy and efficient to loop only over the diagonal entries of a quadratic sparse matrix, as shown in Algorithm 7.

Algorithm 7 Looping over diagonal elements in a CRS-matrix with diagonal-ordered columns

```
1: function CRS-LOOP
2:     for i = 0 : m do
3:         diagonalEntry(i) = dI[rI[i]]
4:     end for
5: end function
```

6.1.1 Construction of auxiliary matrix for $H(\mathbf{curl}, \Omega)$ problems

The construction of the auxiliary matrix for $H(\mathrm{curl}, \Omega)$ problems and edge-element discretization has a higher complexity than for the more trivial scalar or vectorial nodal version. Therefore a small example, is provided here.
Assume we have four elements, as shown in Fig. 6.1 with global and element-wise node-numbers.

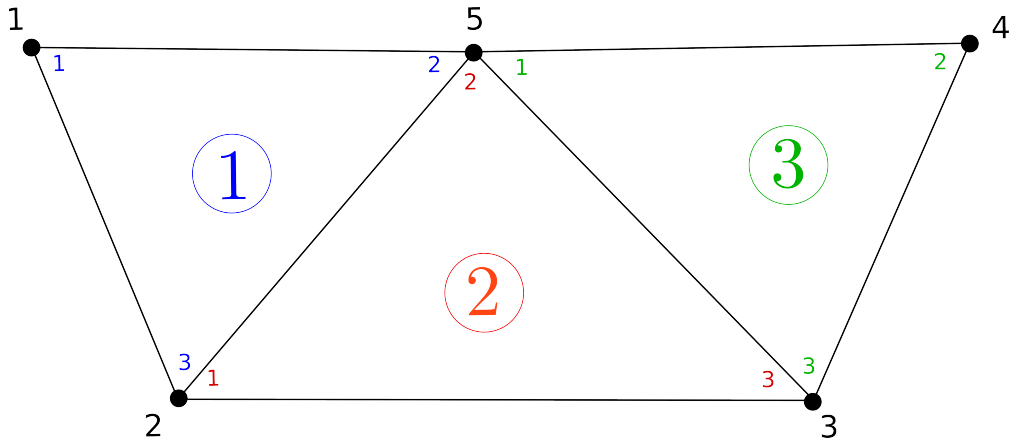

Figure 6.1: Example for the construction of the auxiliary-matrix

According to eq. (4.25), we have to perform an element-wise assembling of contributions into the auxiliary matrix. For element 1, the local auxiliary-matrix $\mathbf{B}^1$ is a (3×3) matrix, since the triangular element has three nodes and looks as follows:

$$\mathbf{B}^1 = \begin{pmatrix} b_{12} + b_{13} & -b_{12} & -b_{13} \\ -b_{21} & b_{21} + b_{23} & -b_{23} \\ -b_{31} & -b_{32} & b_{31} + b_{32} \end{pmatrix}. \tag{6.1}$$

Note that the entries in this matrix represent the distance between the nodes with local node-numbering. Building the element-wise auxiliary-matrices ($\mathbf{B}^2$ and $\mathbf{B}^3$)for the other two elements is straight forward. The next step is to assemble them into the global auxiliary matrix $\mathbf{B}$, using global node-numbers.
This global matrix has a size of (5×5) and the contributions from the different elements are coloured, according to their original element.

$$\mathbf{B} = \begin{pmatrix} b_{12} + b_{15} & -b_{12} & 0 & 0 & -b_{15} \\ -b_{21} & b_{25} + b_{21} + b_{25} + b_{23} & b_{23} & 0 & -b_{25} - b_{25} \\ 0 & -b_{32} & b_{34} + b_{35} + b_{32} & -b_{34} & -b_{35} \\ 0 & 0 & -b_{43} & b_{43} + b_{45} & -b_{45} \\ -b_{51} & -b_{52} - b_{52} & -b_{53} - b_{53} & -b_{54} & b_{54} + b_{35} + b_{35} + b_{25} + b_{51} + b_{52} \end{pmatrix} \tag{6.2}$$

For the actual implementation, the construction of the element-wise matrices can be bypassed, if a construct, called *edgeList* is introduced, which stores for each edge the appropriate global node-numbers (indices in the auxiliary matrix) and the distance between these nodes. Then the assembling can be carried out by looping over all edges, as presented in Algorithm 8.

Algorithm 8 Building the auxiliary-matrix by edge-wise contributions

```
1: function AuxiliaryMatrixEdge
2:     numRows = GetNumNodes() // get number of all nodes in the mesh
3:     for i = 0 : numRows do
4:         edges = edgeList[i] // array containing edges, connected to node i
5:         for j = 0 : edges.size() do
6:             nodes = edges[j].nodes() // array containing the two nodes of edge j
7:             invDist = 1.0/edges[j].dist() // inverse distance between two nodes
8:             for k = 0 : 2 do
9:                 // Fill diagonal entries in B
10:                B[nodes[k], nodes[k]] = invDist
11:                // Fill off-diagonals in in B
12:                l = (k == 0)?  1 : 0
13:                B[nodes[k], nodes[l]] = −invDist
14:            end for
15:        end for
16:    end for
17: end function
```

6.2 Smoother Algorithms

The smoothers-classes in CFS++ consist of two main methods: a *setup*- and a *solve*-phase. In the first, all results, which are needed for the computation and do not change over the iterations, e.g. diagonal inverse or patches, are stored here. This has the benefit, that it is known in the fast solve-step. In the following sections, the generic system

$$\mathbf{K} \cdot \underline{u} = \underline{f}$$

will be solved for $\mathbf{K} \in \mathbb{R}^{N \times N}$, using damped-Jacobi or Arnold-Falk-Winther (AFW) smoother.

6.2.1 Damped Jacobi

As already presented in Section 2.1.1, the damped Jacobi method reads as

$$\underline{u}^{\nu+1} = \underline{u}^{\nu} + \omega \mathbf{D}^{-1} \left(\underline{f} - \mathbf{K} \cdot \underline{u}^{\nu} \right), \quad \text{for} \quad \nu = 0, 1, ...$$

The only candidate for the setup-phase is the inversion of the diagonal element, stored in array *invDiag*.
The algorithm for the solve-phase is straight forward, as presented in Algorithm 9.

Algorithm 9 Damped-Jacobi solve-step with previously computed inverse diagonals

1: **function** DAMPEDJACOBISOLVE
2: **for** $\nu = 1 : smoothSteps$ **do**
3: $\underline{r}^{\nu} = \underline{f} - \mathbf{K} \cdot \underline{u}^{\nu}$
4: **for** $i = 0 : N$ **do**
5: $u^{\nu+1}[i] = u^{\nu}[i] + \Omega \cdot r[i] \cdot invDiag[i]$
6: **end for**
7: **end for**
8: **end function**

6.2.2 Arnold-Falk-Winther

This type of smoother is more involved than Jacobi- or Gauss Seidl-method, because the important part of the smoother are not the entries in the system matrix but their connections between each other. In Section 4.4 it was mentioned that edges, which are connected to a common node, are smoothed together, by extracting submatrices out of the global system matrix. The approach of eq. (4.30) is useful for a nice mathematical description but when writing the algorithm, the use of such matrices would undo all performance optimizations before because we would have to perform a double matrix-product for every node of our system. Therefore we split up the initial setup-phase into a method *CreatePatches()* and *ExtractPatches()*, which are described in the following.

Creation of patches: The creation of patches, as depicted in Fig. 6.2 has to be performed for every node of the current hierarchy-level. The main purpose of this method is to find and store indices of the system matrix, which are extracted in the *ExtrachtPatches()* method. This can easily be achieved by two methods. The first method is to loop over every diagonal in the auxiliary-matrix, which contains an entry for every node, and store it's off-diagonal elements, which correspond to the edges, connected to the diagonal point (if the matrix is viewed as an adjacency graph). For example, let us consider node i in Fig. 6.2, which has the position B_{ii} in the auxiliary-matrix $\mathbf{B}$. Node j has a direct path (edge) to this node and therefore it is an off-diagonal B_{ij}. By doing this we can identify the edge e_{ij}, get its index in the system-matrix, store it in the patch of node i and continue our search for neighbours of this node. For the actual implementation, a slightly different approach was used, using previously gathered information about the grid. Therefore we use an edge-map, which stores for each edge, both connected nodes. By looping over every node and identifying every occurence-position of this node in the edge-map[1], we can reduce the complexity because as soon as we obtained the occurence, the index in the system matrix needs no further lookup in other maps.

[1] When using hash-maps (e.g. boost::`unordered_map`), this has on average constant time $\mathcal{O}(1)$, in the worst case $\mathcal{O}(N)$.

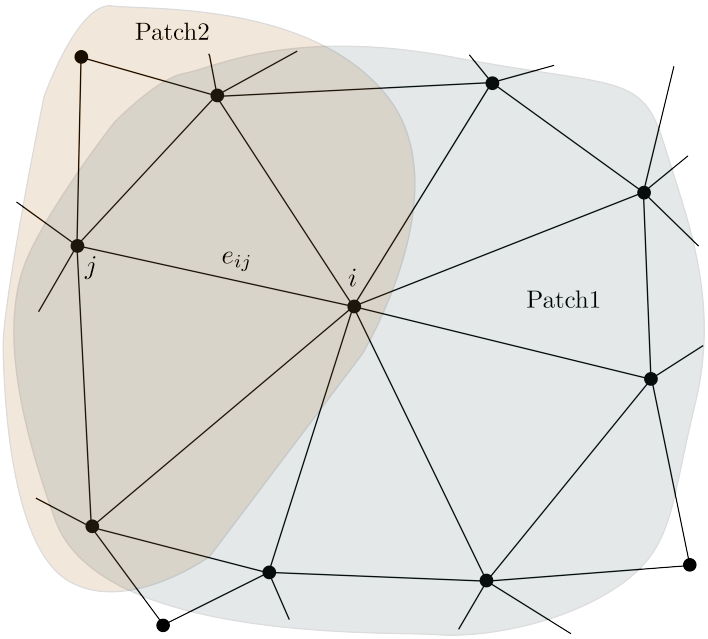

Figure 6.2: Edge patches for Arnold-Falk-Winter smoother

Extraction of patches: This method is needed to improve the performance of the actual solve-step afterwards. *Extraction* in this context means to form small submatrices $\mathbf{K}^j$ from the global system matrix, according to the previously computed edge- (system-matrix)-indices, stored in patches. Also the inversion of these submatrices $(\mathbf{K}^j)^{-1}$ is stored, since they do not change during successive iterations.

Solve-step: Before we analyse the algorithms, let us revisit the basic iteration-formula for the AFW smoother from eq. (4.30)

$$\mathbf{K}^j = \mathbf{R}_j \mathbf{K}_h \mathbf{R}_j^T,$$

$$\underline{u}_{i+1}^j = \underline{u}_i^j + \mathbf{R}_j^T (\mathbf{K}^j)^{-1} \mathbf{R}_j (\underline{f}_h - \mathbf{K} \cdot \underline{u}_i).$$

We can clearly identify the crucial part of the above formula, which is the matrix-vector product $\mathbf{K} \cdot \underline{u}_i$, with $\mathbf{K}$ as the full system-matrix. Because this need to be performed for every node in the solve-step, which is obviously a bad idea, if performance matters. To circumvent this problem, we only update our matrix-vector product for the rows in the matrix, which are contained in the patches for the particular node. The solve-step algorithm is depicted in Algorithm 10.
Another important aspect of this smoother is the overlapping of patches as seen in Fig. 6.2. Without these connections of patches, we end up with a standard point Gauss-Seidel method, which would lead to convergence problems, at least for electromagnetic problems, as shown in Fig. 5.8. To prevent confusion, the patches, defined above are, not at all, the agglomerates, needed for the construction of hierarchy-levels, as presented in Sec. 3.2.2. These agglomerates are constructed, so that there is a disjoint splitting of the computational domain, patches on the other hand must have a cut set, otherwise the convergence is not assured. A possible (heuristical) explanation might be that the information for nodal Lagrangian ansatz functions is located at discrete nodes, with no additional condition for the interaction between nodes,

Algorithm 10 Solve step for the Arnold-Falk-Winther smoother

function AFWSTEP
  for $n = 0 :$ nodes **do**
    $\underline{p} \leftarrow GetPatchOfNode[n]$
    $\underline{e}_{rhs} \leftarrow GetRHSofPatch[n]$, $\underline{e}_{sol} \leftarrow GetSolutionOfPatch[n]$ //extract from global vectors
    for $i = 0 : \underline{p}.size()$ **do**
      $\mathbf{K}_{rP} \leftarrow ExtractRowFromK(p[i])$
      $r_S[i] = \mathbf{K}_{rP} \cdot \underline{e}_{sol}$
    end for
    $\underline{tmp1} = \underline{e}_{rhs} - \underline{r}_S$
    $\underline{tmp2} = \underline{e}_{sol} + (\mathbf{K}^j)^{-1} \cdot \underline{tmp1}$
    $\underline{u}^j_{i+1} \leftarrow InsertInCorrectPlace(\underline{tmp2}, \underline{p})$
  end for
end function

because these ansatz functions (composed of Lagrangian polynomials) have a compact support and therefore do not exchange information between each other. For edge-elements it is more involved, since not only the kind of polynomials for the ansatz function is different (integrated Legendre polynomials) but also the matching of these functions at connecting nodes, in 3D also face, has to be ensured, so there are definitely conditions between edges, which must be satisfied and this kind of smoother takes care of these.

Chapter 7

Summary

The aim of this work was to implement algebraic multigrid methods in CFS++, in order to be able to solve equations arising from elliptic partial differential equations in $H^1(\Omega), (H^1(\Omega)^d$ and $H(\text{curl}, \Omega)$ space and to apply these to solve problems in different physical fields.
The first part was a thorough literature research, combined with small primal implementations in Matlab were carried out, in order to understand the basic principles. Extending this framework, resulted in an AMG-framework in Matlab, which uses the exported linear system and some additional informations from CFS++ and solves the problem, using AMG-methods. This additional information consists mainly of geometry-data, which is needed for the auxiliary-based AMG-approach. If the problem shall be solved via a standard AMG-solver, no additional information, regarding the geometry is needed, because it uses the underlying graph of the system matrix as a virtual mesh.
With this framework it was possible to test different approaches and algorithms, in order to find out the most promising approach to implement in CFS++. For the Matlab-part, only a standalone solver (no preconditioner) was implemented, using V- and W-cycles. In order to find out the best approach, a convergence comparison between the standalone-solver and Matlab-intern GMRES and PCG were carried out for three generic physical fields (electrostatic, 3D mechanic and electromagnetic). This comparison showed promising results with regard to convergence. In all three physical fields, AMG was able to outperform the other two solvers also with respect to execution time.
In the next step, the previously tested algorithms were implemented in CFS++ with additional performance improvements, especially using CFS++ -intern patterns and graph-structures of sparse matrices. Especially the use of Intel MKL for the double matrix product in the Galerkin-operator, brought significant performance benefits, compared to the self-written sparse matrix-matrix product.
A topic, definitely worth further investigation is the analysis of performance-improvement, when computing this sparse matrix-matrix product in parallel, because for larger systems this can have a significant impact. Regarding the parallelization, another improvement would be a parallel setup-phase, using domain-decomposition and parallel coarsening. The difficulty is that all mentioned coarsening strategies, besides agglomeration, are pure serial. In order to use a parallel strategy, completely different algorithms and distributed storage- and communication-concepts

have to be implemented, as shown by Reitzinger [6].
Especially for the application in acoustics, an extension to higher order finite elements is beneficial and a mathematical challenging task. A next step for the implementation could be, to extend the computation to complex valued matrices, arising for example from the Helmholtz-equation in acoustics or time-harmonic cases in general. The ansatz to solve this problem is shown in [9], where the complex-valued matrix is split into a real and an imaginary part and the transfer operators are computed, based on the coarsening of the auxiliary matrix.

Bibliography

[1] W. Hackbusch. Multigrid Methods and Applications. *Springer Series in Computational Mathematics*, 1985.

[2] W. L. Briggs. A Multigrid Tutorial. *SIAM*, 1987.

[3] G. Haase and U. Langer. Skriptum zur Vorlesung Multigrid-Methoden. *Abteilung Numerische Mathematik und Optimierung, Institut für Analysis und Numerik, Johannes Kepler Universität Linz*, 1998.

[4] K. Stüben. Algebraic Multigrid (AMG): An Introduction with Applications. *German National Research for Information Technology, Institute for Algorithms and Scientific Computing SCAI, St. Augustin Germany*, 1999.

[5] C.R. Johnson. Inverse M-matrices. In *Linear Algebra and its Applications*, volume 47, pages 195 – 216, October 1982.

[6] S. Reitzinger. Algebraic Multigrid Methods for Large Scale Finite Element Equations. *Ph.D. thesis, Institut für Analysis und Numerik, Johannes Kepler Universität Linz*, 2001.

[7] R. Hiptmair. Multigrid Methods for Maxwell's Equations. *SIAM Vol. 36, No.1, pp. 204-225*, 1998.

[8] J. Mandel, M. Brezina, and P. Vanek. Energy Optimization of Algebraic Multigrid Basis. *UCD/CCM report 125*, 1998.

[9] M. Kaltenbacher. Numerical Simulation of Mechatronic Sensors and Actuators: Finite Elements for Computational Multiphysics. *Springer Berlin Heidelberg*, 2015.

[10] R. Beck. Algebraic multigrid by component splitting for edge elements on simplicial triangulations. *Preprint SC 99-40, Konrad-Zuse-Zentrum für Informationstechnik Berlin*, 1999.

[11] M. Kaltenbacher. Advanced Simulation Tool for the Design of Sensors and Actuators. In *Procedia Engineering, Proc. Eurosensors XXIV, Linz, Austria*, volume 5, pages 597 – 600, September 2010.

[12] M. Abramowitz and I. Stegun. Handbook of Mathematical Functions: with Formulas, Graphs and Mathematical Tables . 1965.

[13] M. Kaltenbacher. Computational Acoustics. *CISM International Centre for Mechanical Sciences, Chapter Direct and Iterative Solvers by U. Langer and M. Neumüller*, 2018.

[14] M. Wagner. Algebraic Multigrid Methods on Parallel Architectures. *Diplomarbeit, Institut für Mikroelektronik, TU Wien*.

[15] M. Jung, U. Langer, A. Meyer, W. Queck, and M. Schneider. Multigrid Preconditioners and their Application. In *Proceedings of the 3rd GDR Multigrid Seminar held at Biesenthal, Karl- Weierstraß Institut für Mathematik*, pages 11 – 52, 1989.

[16] S. Zaglmayr. High Order Finite Element Methods for Electromagnetic Field Computation . In *Ph.D. thesis, Johannes Kepler University, Linz*, 2006.

List of Figures

2.1 Different preconditioners for CG solver, only MG preconditioner shows scalability 6
2.2 Random initial solution $\underline{u}^0$ . . . 8
2.3 Solution $\underline{u}^5$ after 5 steps with non-optimal $\omega = 1$ for damped Jacobi . . . 8
2.4 Solution $\underline{u}^5$ after 5 steps with optimal $\omega = \frac{2}{3}$ for damped Jacobi . . . 8
2.5 Solution $\underline{u}^5$ after 15 steps with non-optimal $\omega = 1$ for damped Jacobi . . . 8
2.6 Solution $\underline{u}^5$ after 15 steps with optimal $\omega = \frac{2}{3}$ for damped Jacobi . . . 8
2.7 Solution $\underline{u}^5$ after 100 steps with non-optimal $\omega = 1$ for damped Jacobi . . . 9
2.8 Solution $\underline{u}^5$ after 100 steps with optimal $\omega = \frac{2}{3}$ for damped Jacobi . . . 9
2.9 Error after some smoothing steps . . . 12
2.10 smooth error restricted to the coarse mesh . . . 12
2.11 Restriction by injection [6] . . . 13
2.12 Restriction by weighting [6] . . . 13
2.13 Grid hierarchy, needed for V- or W-cycles [4] . . . 15
2.14 Different cycles, depending on the cycle-index γ [13] . . . 16

3.1 Left picture shows the error after some smoothing steps and the right one is the coarsened grid with coarsening along strong connections [14] . . . 19
3.2 Ruge-Stüben coarsening [3], with λ_i-values for undecided points, black color for coarse- and white for fine-grid points . . . 22
3.3 Agglomerates [6] . . . 23

4.1 FE-mesh [6] . . . 26
4.2 Convergence Comparison of scalar elliptic problem . . . 29
4.3 Node-connections in a FE-mesh . . . 31
4.4 Convergence Comparison for static Navier's equation . . . 33
4.5 Convergence comparison for electromagnetic problem . . . 36

5.1 Capacitor setup . . . 37
5.2 Convergence comparison for the 2D capacitor problem . . . 39
5.3 Loaded beam . . . 40
5.4 Convergence comparison between AMG-CG, GMRES and CG . . . 43
5.5 Convergence comparison using AMG-CG for different mesh distortions . . . 44
5.6 Current loaded coil . . . 45
5.7 Convergence comparison between AMG-CG, GMRES and CG for the current loaded coil . . . 48

5.8 Convergence comparison with a wrong smoother 49

6.1 Example for the construction of the auxiliary-matrix 52
6.2 Edge patches for Arnold-Falk-Winter smoother 55

List of Tables

5.1 Wallclock-times of different system-sizes 40
5.2 Wallclock-times for different system-sizes 44
5.3 Wallclock-times for different system-sizes 48

List of Algorithms

1 Coarse Grid Correction 13
2 Two Grid Cycle 15
3 V- and W-Cycle (Multigrid Cycle) 15
4 Standard Coarsening 22
5 Preconditioned Conjugate Gradient Method 23
6 Looping over every non-zero element in a CRS-matrix 51
7 Looping over diagonal elements in a CRS-matrix with diagonal-ordered columns 51
8 Building the auxiliary-matrix by edge-wise contributions 53
9 Damped-Jacobi solve-step with previously computed inverse diagonals 54
10 Solve step for the Arnold-Falk-Winther smoother 56