FRIEDRICH-ALEXANDER-UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT • DEPARTMENT INFORMATIK

Lehrstuhl für Informatik 10 (Systemsimulation)

Multigrid in $H(\mathbf{curl})$ on Hybrid Tetrahedral Grids

Daniel Bauer

Master's Thesis

Multigrid in $\boldsymbol{H}(\mathbf{curl})$ on Hybrid Tetrahedral Grids

Daniel Bauer

Master's Thesis

Aufgabensteller:	Prof. Dr. Ulrich Rüde
Betreuer:	Nils Kohl, M. Sc.
Bearbeitungszeitraum:	08.08.2022 – 08.02.2023

Erklärung:

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Der Universität Erlangen-Nürnberg, vertreten durch den Lehrstuhl für Systemsimulation (Informatik 10), wird für Zwecke der Forschung und Lehre ein einfaches, kostenloses, zeitlich und örtlich unbeschränktes Nutzungsrecht an den Arbeitsergebnissen der Master's Thesis einschließlich etwaiger Schutzrechte und Urheberrechte eingeräumt.

Erlangen, den 06.02.2023 .....

Abstract

This thesis is concerned with the efficient and scalable solution of **curl-curl** type partial differential equations using matrix-free multigrid. After showing how **curl-curl** problems arise from Maxwell's equations, we introduce finite element exterior calculus to acquaint the reader with structure preserving discretization and the Hodge decomposition. Based on these concepts we choose as discretization linear Nédélec edge elements of the first kind and continuous, piecewise linear Lagrangian polynomials to represent scalar potentials. Moreover, we introduce Hiptmair's hybrid smoother which makes multigrid applicable to problems in $\boldsymbol{H}(\mathbf{curl})$. The pivotal idea is to smooth both in the Nédélec space and the space of scalar potentials. We use Chebyshev and Gauss-Seidel smoothers for these sub-problems, respectively.

The main contribution of our work is the implementation of Nédélec edge elements and the hybrid smoother in the finite element framework *HyTeG*. Code generation techniques are used to derive compute kernels of (bi-)linear forms and grid transfer operators automatically. Numerical experiments show that our code attains the optimal (quadratic) convergence rate of the L^2 discretization error. Furthermore, we observe that it is robust with respect to the choice of coefficients and applicable to more complex domains than the theory guarantees. Moreover, the number of V-cycles needed to achieve a certain residual reduction is nearly constant.

Zusammenfassung

Gegenstand dieser Arbeit ist die effiziente und skalierbare Lösung von partiellen Differenzialgleichungen der **curl-curl**-Art mithilfe matrixfreier Mehrgitterverfahren. Zuerst wird gezeigt wie sich **curl-curl**-Probleme aus den Maxwellschen Gleichungen herleiten lassen. Es folgt eine Einführung in finite element exterior calculus, um den Leser mit strukturerhaltenden Diskretisierungen und der Hodge-Zerlegung vertraut zu machen. Diese Konzepte führen zu unserer Wahl der Finiten Elemente Diskretisierung, den linearen Nédélec Kantenelementen erster Art und stetigen stückweise linearen Lagrange Polynomen zur Darstellung von Skalarpotenzialen. Darüber hinaus stellen wir Hiptmairs Hybridglätter vor, welcher Mehrgitterverfahren für Probleme in $\boldsymbol{H}(\mathbf{curl})$ anwendbar macht. Die ausschlaggebende Idee ist es, sowohl im Nédélec Raum als auch im Raum der Skalarpotenziale zu glätten. Für die jeweiligen Subprobleme benutzen wir Chebyshev und Gauß-Seidel Glätter.

Der entscheidende Beitrag dieser Arbeit ist die Implementierung der Nédélec Kantenelemente und des Hybridglätters im Finite Elemente Framework *HyTeG*. Dazu verwenden wir Code-Generierung, um Routinen zur Integration von (Bi-)linearformen und Gittertransfer-Operatoren automatisch herzuleiten. Zuletzt stellen wir numerische Experimente vor, die zeigen, dass unsere Implementierung den L^2-Diskretisierungsfehler mit der optimalen (quadratischen) Rate reduziert. Des Weiteren wird gezeigt, dass die Implementierung robust gegenüber der Wahl von Koeffizienten ist und auf allgemeinere Gebiete anwendbar ist als die Theorie garantiert. Darüber hinaus ist die Anzahl der V-Zyklen, die benötigt werden, um eine gegebene Residuumsreduktion zu erzielen, beinahe unabhängig von der Gitterweite.

Contents

Glossary

CG	conjugate gradient 26, 49
CSE	common subexpression elimination 39
DoF	degree of freedom 17–20, 26, 33–36, 38, 40–42, 44–46, 51, 54, 58
FEEC	finite element exterior calculus vii, ix, 1–3, 5, 7, 8, 12, 14, 16, 18, 57
FEM	finite element method 1, 3, 5, 7, 8, 14, 24, 58
LSE	linear system of equations 1, 5, 7, 21, 37, 40
NHR@FAU	Erlangen National High Performance Computing Center 55
PDE	partial differential equation vii, 1, 2, 5–7, 10, 21, 50, 52
RHS	right-hand side 3, 24, 26, 30, 40, 47, 49, 52

Chapter 1

Introduction

Many applications require large scale simulations of electromagnetic fields. One example is the development of nuclear fusion reactors. These devices are fueled by plasma, which is heated to over 100 million degrees Celsius, causing electrons to separate from the nuclei. Due to the extreme heat, the plasma can only be confined by the use of electromagnetic fields [1]. As nuclear fusion would be a clean and safe source of energy, it has the potential to play a valuable role in our battle against global warming [2].

An essential component in electromagnetic simulations is a solver for Maxwell's equations. Like solving other partial differential equations (PDEs) finding a solution to Maxwell's system comprises two steps: discretizing the continuous problem and solving the resulting linear system of equations (LSE). As demands on the resolution of the simulation and complexity of the domain increase, these systems typically get quite large. As a result, high performance computing is an indispensable tool supporting a wide range of domain sciences. Therefore, it is crucial to devise specialized algorithms that scale well on massively parallel architectures. By this we mean that the computational effort does not grow faster than the number of unknowns. One of the few algorithms that can achieve this property is multigrid [3, p. 20].

A popular discretization method is the finite element method (FEM). However, it is an intricate problem to develop stable finite element discretizations for Maxwell's equations. Over the past decades, a new theory titled finite element exterior calculus (FEEC) has emerged. It provides general techniques for finding suitable discretizations for a variety of different problems [4]. A key idea is to preserve certain topological and algebraic structures of the continuous problem when going to the discrete level instead of just approximating them [5].

An important mathematical concept in the context of Maxwell's equations is $\boldsymbol{H}(\mathbf{curl})$, the Hilbert space of L^2-integrable functions whose **curl** exists in the weak sense and is also L^2-integrable. For completeness, the **curl** of a three-dimensional vector field $\boldsymbol{u}$ is defined as [6, p. 14]

$$\mathbf{curl}\,\boldsymbol{u} = \left(\frac{\partial u_3}{\partial y} - \frac{\partial u_2}{\partial z}, \frac{\partial u_1}{\partial z} - \frac{\partial u_3}{\partial x}, \frac{\partial u_2}{\partial x} - \frac{\partial u_1}{\partial y}\right)^T . \tag{1.1}$$

FEEC explains why Nédélec edge elements [7], which are conforming in $\boldsymbol{H}(\mathbf{curl})$, are the discretization of choice for our setting. However, standard multigrid schemes are not effective for problems in $\boldsymbol{H}(\mathbf{curl})$. The development of efficient multigrid schemes for Maxwell's equations has been pioneered by Hiptmair [8].

This work brings the theory of FEEC and Hiptmair's algorithm to the highly scalable finite element software framework *HyTeG*[1] [9]. The pivotal idea of *HyTeG* is to implement matrix-free multigrid on hybrid tetrahedral grids. These special block-structured grids combine the flexibility of unstructured topology with the performance benefits of uniform refinement [9].

In this thesis we make use of the above concepts to solve the model problem

$$\alpha \,\mathbf{curl}\,\mathbf{curl}\,\boldsymbol{u} + \beta \boldsymbol{u} = \boldsymbol{f} \quad \text{in } \Omega\,, \tag{1.2a}$$

$$\boldsymbol{u} \times \boldsymbol{n} = 0 \quad \text{on } \partial\Omega\,, \tag{1.2b}$$

where $\boldsymbol{f} \in \boldsymbol{L}^2(\Omega)$, α and β are positive constants and $\Omega \subset \mathbb{R}^3$ denotes the domain. Throughout the text, vectors as well as vector-valued functions, function spaces and operators are printed in boldface.

The PDE (1.2) arises from Maxwell's equations [8]

$$\epsilon \frac{d}{dt}\boldsymbol{E} + \sigma \boldsymbol{E} - \mathbf{curl}\,\boldsymbol{H} = \boldsymbol{j}_i \quad \text{in } \Omega \times (0,T)\,, \tag{1.3a}$$

$$\mu \frac{d}{dt}\boldsymbol{H} + \mathbf{curl}\,\boldsymbol{E} = 0 \quad \text{in } \Omega \times (0,T)\,, \tag{1.3b}$$

with electric field $\boldsymbol{E}(\boldsymbol{x},t)$, magnetic field $\boldsymbol{H}(\boldsymbol{x},t)$, dielectric constant $\epsilon(\boldsymbol{x})$, magnetic permeability $\mu(\boldsymbol{x})$, conductivity $\sigma(\boldsymbol{x})$ and intrinsic current $\boldsymbol{j}_i(\boldsymbol{x},t)$. It is required that $\epsilon, \mu \in L^\infty(\Omega)$ are uniformly positive and $\sigma \in L^\infty(\Omega)$ is non-negative almost everywhere.

In this setting, homogeneous tangential boundary conditions

$$\boldsymbol{E} \times \boldsymbol{n} = 0 \quad \text{on } \partial\Omega \tag{1.4}$$

model perfectly conducting walls [8].

The system of PDEs (1.3) can be transformed into a single second order problem by inserting the second equation into the time-derivative of the first to eliminate the magnetic field $\boldsymbol{H}$ [8]:

$$\epsilon \frac{d^2}{dt^2}\boldsymbol{E} + \sigma \frac{d}{dt}\boldsymbol{E} + \mathbf{curl}\left(\frac{1}{\mu}\,\mathbf{curl}\,\boldsymbol{E}\right) = \frac{d}{dt}\boldsymbol{j}_i \quad \text{in } \Omega \times (0,T)\,, \tag{1.5}$$

where initial conditions for $\boldsymbol{E}$ and $\frac{d}{dt}\boldsymbol{E}$ must be provided. If we finally discretize (1.5) in time, e.g. with the Crank-Nicolson scheme, each time step comprises solving the

[1] Available at `https://i10git.cs.fau.de/hyteg/hyteg`.

system [8]

$$\frac{1}{4}\Delta t^2 \,\mathbf{curl}\left(\frac{1}{\mu}\,\mathbf{curl}\,\boldsymbol{E}\right) + \left(\epsilon + \frac{1}{2}\sigma\Delta t\right)\boldsymbol{E} = \boldsymbol{f} \quad \text{in } \Omega\,, \tag{1.6a}$$

$$\boldsymbol{E} \times \boldsymbol{n} = 0 \quad \text{on } \partial\Omega\,. \tag{1.6b}$$

Here, Δt is the positive time step size and the right-hand side (RHS) $\boldsymbol{f}$ depends on Δt, $\boldsymbol{j}_i$ and $\boldsymbol{E}$ in the previous two time steps. By assuming that all coefficients are constant, we arrive at our model problem (1.2).

This work is organized as follows: In Chapter 2 we give a short introduction to FEM and summarize the fundamentals of FEEC. Furthermore, we introduce our chosen discretization, linear Nédélec elements of the first type. It follows in Chapter 3 a presentation of the multigrid algorithm and a discussion of the Chebyshev smoother. Then, we familiarize the reader with Hiptmair's hybrid smoother and show that it clearly outperforms standard multigrid smoothers. Chapter 4 introduces *HyTeG* and details issues arising in the implementation. Next, we perform numerical experiments to assess the correctness and efficiency of our code in Chapter 5. Lastly, we conclude the thesis and give an outlook onto possible future work.

Chapter 2

Structure Preserving Finite Element Discretization

The following two chapters provide the reader with the theoretical background relevant for the solution of (1.2). Since many fields come together, we give an overview first. Partial differential equations (PDEs) like (1.2) are continuous by nature. Since computers are limited to discrete and finite representations of functions and numbers, a PDE must be discretized first so that it can be solved numerically. Our method of choice is the finite element method (FEM). Over time, a plethora of finite element discretization schemes has been developed. Although the canonical choice for many problems are piecewise Lagrangian polynomial elements, the appropriate scheme for our equation are Nédélec elements for $\boldsymbol{H}(\mathbf{curl})$. Throughout the text we denote the space of continuous, piecewise linear Lagrangian polynomials as $\mathcal{P}_1^- \Lambda^0$ and Nédélec-elements of type I and order 1 as $\mathcal{P}_1^- \Lambda^1$. The reasons for this choice of nomenclature become clear below. Why the Nédélec space is suitable is explained by finite element exterior calculus (FEEC), a theory for finding convergent and stable discretizations for a wide class of problems by preserving structure of the continuous problem exactly.

This chapter presents the main ideas of the FEM in general, gives an introduction to FEEC and familiarizes the reader with Nédélec finite elements. At the same time, it lays the theoretical foundations for the multigrid iteration which we use to solve the linear system of equations (LSE) resulting from the discretization in an efficient and scalable way. An introduction to general multigrid theory and Hiptmair's [8] hybrid smoother for $\boldsymbol{H}(\mathbf{curl})$ is presented in Chapter 3.

2.1 Finite Element Method

The FEM is a special case of the Galerkin method which is concerned with finding the solution $u \in V$ of variational equations of the form [10, p. 100]

$$a(u, v) = b(v) \quad \forall v \in V\,. \tag{2.1}$$

Here, V is a Hilbert space with associated dual space V', the space of linear continuous functionals $b : V \to \mathbb{R}$ [10, pp. 77 sq.]. Moreover, $b \in V'$ is a linear functional and $a(\cdot, \cdot) : V \times V \to \mathbb{R}$ is a bilinear form.

Many boundary-value problems of PDEs can be cast into the form (2.1). This is usually done by integrating the strong form of the differential equation over the domain Ω, multiplying by a test function $v \in V$ and applying partial integration. For example, our model problem (1.2) has the equivalent weak formulation [8]:

Find $\boldsymbol{u} \in \mathring{\boldsymbol{H}}(\mathbf{curl})$ such that for all $\boldsymbol{v} \in \mathring{\boldsymbol{H}}(\mathbf{curl})$

$$\int_\Omega \alpha\, \mathbf{curl}\, \boldsymbol{u} \cdot \mathbf{curl}\, \boldsymbol{v} + \int_\Omega \beta \boldsymbol{u} \cdot \boldsymbol{v} = \int_\Omega \boldsymbol{f} \cdot \boldsymbol{v}\,, \tag{2.2}$$

where $\boldsymbol{f} \in \boldsymbol{L}^2(\Omega)$ and $\alpha, \beta > 0$. In addition, $\mathring{\boldsymbol{H}}(\mathbf{curl})$ is defined by [8]

$$\mathring{\boldsymbol{H}}(\mathbf{curl}) := \left\{ \boldsymbol{u} \in \boldsymbol{L}^2(\Omega) \mid \mathbf{curl}\, \boldsymbol{u} \in \boldsymbol{L}^2(\Omega), \boldsymbol{u} \times \boldsymbol{n} = 0 \text{ on } \partial\Omega \right\}, \tag{2.3}$$

with normal vector $\boldsymbol{n}$ on the boundary $\partial\Omega$. We see that (2.2) attains the form (2.1), by setting $V := \mathring{\boldsymbol{H}}(\mathbf{curl})$, $a(\boldsymbol{u}, \boldsymbol{v}) := \int_\Omega \alpha\, \mathbf{curl}\, \boldsymbol{u} \cdot \mathbf{curl}\, \boldsymbol{v} + \int_\Omega \beta \boldsymbol{u} \cdot \boldsymbol{v}$ and $b(\boldsymbol{v}) := \int_\Omega \boldsymbol{f} \cdot \boldsymbol{v}$.

To establish the existence of a unique solution to the variational equation (2.1), we require that $a(\cdot, \cdot)$ is

1. *continuous*, i.e. it exists a constant $M > 0$ such that

$$|a(u, v)| \leq M \|u\| \|v\| \quad \forall u, v \in V \tag{2.4}$$

2. and V*-elliptic*, i.e. it exists a $\gamma > 0$ such that

$$a(u, v) \geq \gamma \|u\|^2 \quad \forall u \in V\,. \tag{2.5}$$

Under these assumptions, the Lax-Milgram-Lemma [10, Lemma 3.6] states that (2.1) has a unique solution for all $b \in V'$.

To solve (2.1) numerically, the space V is approximated by a finite dimensional subspace $V_h \subset V$, with $n := \dim V_h$. Since V_h is chosen such that it, too, is a Hilbert space with the same inner product as V, the discrete variational equation

$$a(u_h, v_h) = b(v_h) \quad \forall v_h \in V_h \tag{2.6}$$

also has a unique solution $u_h \in V_h$. Solving (2.6) to obtain an approximate solution of the continuous problem (2.1) is known as Galerkin method [10, p. 96]. Céa's Lemma [10, Theorem 3.4] gives an estimate of the error made by this approximation:

$$\|u - u_h\| \leq \frac{M}{\gamma} \inf_{v_h \in V_h} \|u - v_h\|\,. \tag{2.7}$$

The benefit of restricting (2.6) to V_h is that each $u_h \in V_h$ can be represented as a finite dimensional vector $\boldsymbol{s} \in \mathbb{R}^n$ with respect to some basis $\{\phi_i\}$ of V_h according to [10, p. 98]

$$u_h(\boldsymbol{x}) = \sum_{i=1}^{n} s_i \phi_i(\boldsymbol{x}), \quad \boldsymbol{x} \in \Omega\,. \tag{2.8}$$

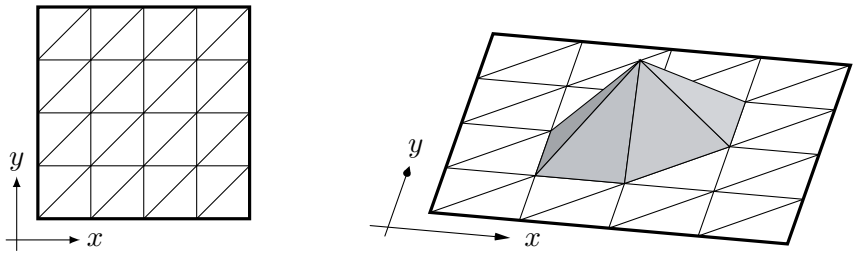
Consequently, (2.6) can be rewritten as the LSE

$$\sum_{j=1}^{n} a(\phi_j, \phi_i) s_j = f(\phi_i), \quad i = 1 \dots n \tag{2.9}$$

which is regular under the assumption of V-ellipticity [10, pp. 98 sq.]. This means that by approximating V by V_h, we reduced the task of solving (2.1) and the associated PDE to solving an LSE. A presentation of the multigrid algorithm that we use for this purpose is given in the following chapter. The main question that remains to be answered is how to choose V_h.

The approach of the FEM is to divide the domain Ω into geometrically simple elements. The resulting *triangulation* with mesh width h is commonly denoted by $\mathcal{T}_h$. Typically, the elements are triangles or rectangles in 2D and tetrahedra or hexahedra in 3D. The basis functions ϕ_i of V_h are usually piecewise polynomials and chosen in such a way that they have a small support, i.e. vanish on most elements. This has the benefit that most coefficients of the LSE (2.9) are zero and the system can be solved efficiently.

The canonical choice of basis functions are continuous, piecewise linear Lagrangian polynomials which span the space $\mathcal{P}_1^- \Lambda^0(\mathcal{T}_h)$. An example is depicted in Figure 2.1. A square domain is subdivided regularly into triangles and a basis function is associated to each node of the triangulation. The basis functions have value 1 at their respective node and are 0 at the other nodes. A sketch of the basis function of the central node is depicted in Figure 2.1b.



(a) Regular triangulation. (b) Basis function of the central node.

Figure 2.1: A square domain subdivided into regular triangles and one of the piecewise linear Lagrangian basis functions.

2.2 Finite Element Exterior Calculus

The $\mathcal{P}_1^- \Lambda^0$ space is a special case of a general finite element space explained by FEEC. The goal of this section is to introduce the reader to the core principles of FEEC and explain why $\mathcal{P}_1^- \Lambda^0$ is not the best choice to solve (1.2). Furthermore, Nédélec edge elements of type I are introduced as another special case of the same general theory,

and it is elucidated why Nédélec elements are the preferred way to solve our problem. This section is mainly based on [4].

FEEC emerged over the course of the past two decades as a new theory to develop and analyze FEMs [4, pp. x, 1]. It aims at finding stable discretizations by preserving certain structure of the continuous problem exactly, not approximately, when going to the discrete level.

2.2.1 Differential Forms

The theory achieves a great deal of generality by relying on the formalism of exterior calculus. In particular, instead of functions, differential forms are considered. Therefore, our first objective is to introduce differential forms and explain how they can be used to represent functions.

Algebraic k-Forms

An (algebraic) k-form ω is an alternating multilinear function mapping k vectors to $\mathbb{R}$ [11, p. 38]. Multilinear is to say linear in each argument and alternating means that the sign of the function changes when two arguments are swapped:

$$\omega(\boldsymbol{v}_1, \dots, \boldsymbol{v}_i, \dots, \boldsymbol{v}_j, \dots, \boldsymbol{v}_k) = -\omega(\boldsymbol{v}_1, \dots, \boldsymbol{v}_j, \dots, \boldsymbol{v}_i, \dots, \boldsymbol{v}_k)\,. \tag{2.10}$$

More formally, we can define the vector space of k-forms [4, pp. 63 sq.]:

Definition 2.1 (k-form, $\mathrm{Alt}^k V$)**.** Let V be an n-dimensional vector space and $k \geq 0$. An alternating multilinear mapping $\omega : \underbrace{V \times \cdots \times V}_{k \text{ times}} \to \mathbb{R}$ is called k-form. The space of k-forms is denoted by $\mathrm{Alt}^k V$. For $k = 0$ we define $\mathrm{Alt}^0 V := \mathbb{R}$. For $k > n$, $\mathrm{Alt}^k V = 0$. Note that $\mathrm{Alt}^1 V$ coincides with the dual space V'.

Two forms of degree k and l can be multiplied by the *wedge product* to give a form of degree $k + l$. While the definition below seems unwieldy, it ensures that the result is again alternating and multilinear. For example, simple pointwise multiplication destroys linearity [11, p. 29].

Definition 2.2 (Wedge product [4, pp. 64 sq.])**.** Let $\omega \in \mathrm{Alt}^k V$ and $\nu \in \mathrm{Alt}^l V$ be two forms of degree k and l, respectively. Their wedge product $\omega \wedge \nu \in \mathrm{Alt}^{k+l}$ is defined by

$$(\omega \wedge \nu)(\boldsymbol{v}_1, \dots, \boldsymbol{v}_{k+l}) = \sum_{\sigma} \operatorname{sign}(\sigma) \omega(\boldsymbol{v}_{\sigma_1}, \dots, \boldsymbol{v}_{\sigma_k}) \nu(\boldsymbol{v}_{\sigma_{k+1}}, \dots, \boldsymbol{v}_{\sigma_{k+l}})\,, \tag{2.11}$$

where we sum over all permutations σ of the integers $\{1, \dots, k+l\}$ with $\sigma_1 < \cdots < \sigma_k$ and $\sigma_{k+1} < \cdots < \sigma_{k+l}$. $\operatorname{sign}(\sigma)$ is $+1$ for an even and -1 for an odd number of permutations [12].

The wedge product is anti-commutative [13], i.e.

$$\omega \wedge \nu = (-1)^{kl} \nu \wedge \omega, \quad \omega \in \mathrm{Alt}^k V, \nu \in \mathrm{Alt}^l V\,. \tag{2.12}$$

Furthermore, it lets us define a basis for $\mathrm{Alt}^k V$. Every basis $\boldsymbol{u}_i, \dots, \boldsymbol{u}_n$ of V induces a basis $u^i, \dots, u^n$ of the dual space $V' = \mathrm{Alt}^1 V$. The basis functions are determined by $u^i(\boldsymbol{u}_j) = \delta_{ij}$ [13]. Moreover, if σ, ρ are increasing index sets ($1 \leq \sigma_1 < \cdots < \sigma_k \leq n$ and analogously for ρ), it holds that

$$u^{\sigma_1} \wedge \cdots \wedge u^{\sigma_k}(\boldsymbol{u}_{\rho_1}, \dots, \boldsymbol{u}_{\rho_n}) = \begin{cases} 1, & \text{if } \sigma = \rho, \\ 0, & \text{else.} \end{cases} \tag{2.13}$$

The number of distinct increasing index sets is $\binom{n}{k}$ and is the same as $\dim \mathrm{Alt}^k V$. Because an algebraic k-form ω is uniquely determined by its values $\omega(\boldsymbol{u}_{\rho_1}, \dots, \boldsymbol{u}_{\rho_k})$, the k-forms $u^{\sigma_1} \wedge \cdots \wedge u^{\sigma_k}$ form a basis of $\mathrm{Alt}^k V$ [13].

Example: k-Forms on $\mathbb{R}^3$

To make the definitions above seem less abstract, we apply them to the case $V = \mathbb{R}^3$. $\mathbb{R}^3$ is important because we are interested in solving boundary value problems on domains in $\mathbb{R}^3$. We denote the coordinates of a vector $\boldsymbol{x} \in V$ by x_i and the basis functions of V' by dx^i. The functions dx^i assign to a vector its i-th coordinate: $dx^i(\boldsymbol{x}) = x_i$. The basis representation permits to write every k-form as

$$k = 0: \quad \omega = c\,, \tag{2.14a}$$

$$k = 1: \quad \omega = a_1 dx^1 + a_2 dx^2 + a_3 dx^3\,, \tag{2.14b}$$

$$k = 2: \quad \omega = a_1 dx^2 \wedge dx^3 - a_2 dx^1 \wedge dx^3 + a_3 dx^1 \wedge dx^2\,, \tag{2.14c}$$

$$k = 3: \quad \omega = c dx^1 \wedge dx^2 \wedge dx^3\,, \tag{2.14d}$$

with coefficients $c \in \mathbb{R}$ and $\boldsymbol{a} \in \mathbb{R}^3$. From this representation we see that there is a natural isomorphism between $\mathrm{Alt}^0\mathbb{R}^3$, $\mathrm{Alt}^3\mathbb{R}^3$ and $\mathbb{R}$ ($\omega \leftrightarrow c$) on the one hand and $\mathrm{Alt}^1\mathbb{R}^3$, $\mathrm{Alt}^2\mathbb{R}^3$ and $\mathbb{R}^3$ ($\omega \leftrightarrow \boldsymbol{a}$) on the other hand. The representative elements c and $\boldsymbol{a}$ are known as *scalar/vector proxies* [4, pp. 66 sq.].

In terms of vector proxies, wedge multiplication by a 0-form is scalar multiplication. The product of two 1-forms correlates to the cross-product of their vector-proxies, whereas multiplying a 1- and a 2-form is equivalent to taking the dot-product of their vector-proxies [4, p. 68]. All other combinations raise the form degree beyond 3 and therefore result in 0.

Differential k-Forms

We have seen how algebraic k-forms can be viewed as scalars and vectors. This already brings us halfway to representing scalar and vector fields by differential forms. We

return to generality and introduce differential forms on finite-dimensional manifolds Ω that are smooth enough to exhibit a tangent space. The tangent space of an n-dimensional manifold Ω at point $\boldsymbol{x}$, written $T_{\boldsymbol{x}}\Omega$, consists of all n-dimensional vectors tangent to Ω at point $\boldsymbol{x}$ [14]. Since it is a vector space, we can construct the space $\mathrm{Alt}^k T_{\boldsymbol{x}}\Omega$. As there is no obvious choice for a specific $\boldsymbol{x} \in \Omega$, it comes natural to introduce functions mapping from Ω to $\mathrm{Alt}^k T_{\boldsymbol{x}}\Omega$. If we further require these functions to be differentiable with respect to $\boldsymbol{x}$, they are known as differential forms [11, p. 41].

Definition 2.3 (Differential k-form, Λ^k, [4, p. 69])**.** Let Ω be an n-dimensional manifold with tangent space $T_{\boldsymbol{x}}\Omega$. A differentiable k-form on Ω is a function $\Omega \to \mathrm{Alt}^k T_{\boldsymbol{x}}\Omega$. It can be written as

$$\omega = \sum_{1 \le \sigma_1 < \cdots < \sigma_k \le n} a_\sigma \, dx^{\sigma_1} \wedge \cdots \wedge dx^{\sigma_k} \,, \tag{2.15}$$

where the coefficients $a_\sigma : \Omega \to \mathbb{R}$ are differentiable functions and $dx^{\sigma_1}, \ldots, dx^{\sigma_n}$ is a basis of the dual space of $T_{\boldsymbol{x}}\Omega$. The vector space of all differential k-forms on Ω is written $\Lambda^k(\Omega)$, or Λ^k for short. Moreover, we precede Λ^k with function spaces to restrict the coefficients a_σ. For example, if we write $\mathcal{P}_1\Lambda^k$, we mean differential k-forms with linear polynomial coefficients.

The wedge product of differential forms is defined pointwise [4, p. 70]:

$$(\omega \wedge \nu)_{\boldsymbol{x}} = \omega_{\boldsymbol{x}} \wedge \nu_{\boldsymbol{x}}, \quad \omega \in \Lambda^k, \nu \in \Lambda^l \,. \tag{2.16}$$

Let us take as example $\Omega = \mathbb{R}^3$. In this case, every tangent space $T_{\boldsymbol{x}}\mathbb{R}^3$ can be identified with $\mathbb{R}^3$. Therefore, $\Lambda^k(\mathbb{R}^3)$ can be thought of as the space of functions $\mathbb{R}^3 \to \mathrm{Alt}^k\mathbb{R}^3$. Now we apply the fact that for $k = 0$ and $k = 3$, elements of $\mathrm{Alt}^k\mathbb{R}^3$ can be represented by real scalar values. Therefore, a differential k-form is associated to a function $\mathbb{R}^3 \to \mathbb{R}$, a scalar field. For $k = 1$ and $k = 2$, vector proxies can be used to interpret a differential k-form $\omega \in \Lambda^k(\mathbb{R}^3)$ as a vector field, i.e. a function $\mathbb{R}^3 \to \mathbb{R}^3$.

Exterior Derivative and Sobolev Spaces

The ability to rewrite functions in terms of differential forms can, in the context of PDEs, only be useful if there is a differentiation operator for them. This operator is known as the *exterior derivative* d and it maps k-forms to $(k+1)$-forms: $\mathrm{d}^k : \Lambda^k(\Omega) \to \Lambda^{k+1}(\Omega)$ [4, p. 71]. Often we drop the index k when there is no risk of confusion. Exterior differentiation can be defined on manifolds [4, p. 72] but here we only give a formula for the case that Ω is a domain in $\mathbb{R}^n$.

This restriction allows us to identify every tangent space $T_{\boldsymbol{x}}\Omega$ with $\mathbb{R}^n$. Hence, if $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k \in \mathbb{R}^n$ are fixed vectors, they can be viewed as elements of the tangent space at all points $\boldsymbol{x}$. Then, if $\omega \in \Lambda^k(\Omega)$ is a differential form, we obtain the differentiable

mapping $\boldsymbol{x} \mapsto \omega_{\boldsymbol{x}}(\boldsymbol{v}_1, \dots, \boldsymbol{v}_k)$ from Ω to $\mathbb{R}$. With this we define the exterior derivative

$$\mathrm{d}\omega_{\boldsymbol{x}}(\boldsymbol{v}_1, \dots, \boldsymbol{v}_{k+1}) = \sum_{j=1}^{k+1} (-1)^{j+1} \partial_{\boldsymbol{v}_j} \omega_{\boldsymbol{x}}(\boldsymbol{v}_1, \dots, \hat{\boldsymbol{v}}_j, \dots, \boldsymbol{v}_{k+1}),$$
$$\omega \in \Lambda^k, \boldsymbol{v}_1, \dots, \boldsymbol{v}_{k+1} \in V\,, \quad (2.17)$$

where the hat marks the omission of the j-th argument and $\partial_{\boldsymbol{v}_j}$ denotes the directional derivative in direction $\boldsymbol{v}_j$. It will later become important that $\mathrm{d} \circ \mathrm{d} = 0$ [13].

As with differential forms, we translate the exterior derivative to the language of vector calculus. When identifying a differential 0-form in three-dimensional space with a scalar field, exterior differentiation is just the gradient operator. For 1-forms, d is identified with **curl** and for 2-forms with the divergence [4, p. 71]. At this point, it becomes clear why it is valuable to have two distinct ways to view vector fields.

Having a differentiation operator at hand, we can define Sobolev spaces of differential forms [13]:

$$H\Lambda^k(\Omega) := \left\{ \omega \in L^2\Lambda^k(\Omega) \mid \mathrm{d}^k\omega \in L^2\Lambda^{k+1}(\Omega) \right\}. \quad (2.18)$$

Pullback, Trace and Integration

Next, we describe another essential operation, the *pullback*. The pullback substantially describes a change of variable [6, p. 161]. We make use of it immediately to define trace and integration of differential forms and come back to it later to define finite element spaces on simplices which arise from affine transformations of a reference tetrahedron.

Let $\omega \in \mathrm{Alt}^k W$ be an algebraic k-form and $B : V \to W$ a linear map between finite dimensional vector spaces. Then, the pullback operation $B^* : \mathrm{Alt}^k W \to \mathrm{Alt}^k V$ is given by [4, p. 67]

$$B^*\omega(\boldsymbol{v}_1, \dots, \boldsymbol{v}_k) = \omega(B\boldsymbol{v}_1, \dots, B\boldsymbol{v}_k)\,. \quad (2.19)$$

The pullback of algebraic forms is used to define the pullback of a differential form $\omega \in \Lambda^k(N)$ under a differentiable map $F : M \to N$ between manifolds. Since the derivative of F is a linear map between tangent spaces: $F' : T_{\boldsymbol{x}}M \to T_{F(\boldsymbol{x})}N$, we can define [4, p. 70]:

$$(F^*\omega)_{\boldsymbol{x}} = F'(\boldsymbol{x})^* \omega_{F(\boldsymbol{x})}\,. \quad (2.20)$$

We remark that the pullback commutes with exterior product and derivative [13]:

$$F^*(\omega \wedge \nu) = F^*\omega \wedge F^*\nu\,, \quad (2.21a)$$
$$F^*(\mathrm{d}\omega) = \mathrm{d}(F^*\omega)\,. \quad (2.21b)$$

If M is a submanifold of N, the trace $\mathrm{tr} : \Lambda^k(N) \to \Lambda^k(M)$ is defined as the pullback of the inclusion $M \hookrightarrow N$ [4, p. 70][13].

With the pullback at hand, we can define the integration of differential forms over parameterized surfaces. Let $F : M \subset \mathbb{R}^k \to N \subset \mathbb{R}^n$ be an orientation preserving parameterization of a k-dimensional surface N. We can then integrate a differential k-form ω over N according to [11, p. 56]

$$\int_N \omega = \int_M F^*\omega = \int_M f(x_1, \ldots, x_k) dx^1 \wedge \cdots \wedge dx^k \,, \tag{2.22}$$

where the value of the last integral in (2.22) is obtained by evaluating the Lebesgue integral of f over M [4, p. 73]. The key idea behind this construction is that a change of variables is performed so that integration is over a k-dimensional subset of $\mathbb{R}^k$. This resolves the complications related to integrating over surfaces. A nice property of the integration of differential forms is that it is independent of any metric or measure on the manifold [4, p. 72].

Note that the form degree and the dimension of N must match. For example 0-forms can be integrated (evaluated) at points, while 1- and 2-forms can be integrated over oriented curves and surfaces, respectively [13].

2.2.2 Structure Preserving Discretization

As previously mentioned, FEEC is a theory to find structure preserving discretizations. These have the characteristic that some properties of the continuous problem are preserved exactly when going to the discrete level, not as $h \to 0$. In FEEC the structure that is being preserved are properties of the de Rham complex, one of the objects studied by homological algebra. We thus first introduce some terms from homological algebra, the de Rham complex and some properties of this complex.

Chain Complexes

A (co-)chain complex is a sequence of vector spaces and operators mapping from one space to the next, with the property that repeated operator application yields zero [13]. The more formal definition involves the terms *graded vector space* and *graded map*.

Definition 2.4 (Graded vector space [4, p. 11])**.** A graded vector space V is a vector space expressed as a direct sum of subspaces V_k indexed by $\mathbb{Z}$.

$$V = \bigoplus_{k=-\infty}^{\infty} V_k \tag{2.23}$$

If almost all V_k are zero, V is called *finite*.

Definition 2.5 (Graded map [4, p. 11])**.** A graded map $f : V \to W$ is a linear map between graded vector spaces with $f(V_k) \subset W_{k+p}$ for all k and fixed *degree* p.

Definition 2.6 (Chain complex [4, p. 12], cochain complex [4, p. 16])**.** A chain complex is a graded vector space V equipped with a graded linear map $\partial : V \to V$ of degree -1 which satisfies $\partial \circ \partial = 0$. The operator ∂ is called the *differential* of the chain complex. Chain complexes are usually written in the form

$$\cdots \longrightarrow V_{k+1} \xrightarrow{\partial_{k+1}} V_k \xrightarrow{\partial_k} V_{k-1} \longrightarrow \cdots \,. \tag{2.24}$$

A cochain complex is like a chain complex, only that the differential is of degree $+1$ and is commonly written as d. Moreover, it is customary to use superscripts instead of subscripts for cochain complexes.

$$\cdots \longrightarrow V^{k-1} \xrightarrow{d^{k-1}} V^k \xrightarrow{d^k} V^{k+1} \longrightarrow \cdots \tag{2.25}$$

Sometimes a complex is abbreviated as (V, ∂) or (V, d).

Given a chain complex with differential ∂, elements of the nullspace $\mathfrak{Z}_k$ of ∂_k are called *k-cycles*. Elements of the range $\mathfrak{B}_k$ of ∂_{k+1} are called *k-boundaries*. Note that $\partial \circ \partial = 0$ implies $\mathfrak{B}_k \subset \mathfrak{Z}_k$. Therefore, the quotient space $\mathcal{H}_k := \mathfrak{Z}_k / \mathfrak{B}_k$ is well-defined. It is known as the *k-th homology space*. If $\mathfrak{Z}_k = \mathfrak{B}_k$ for all k, then all homology spaces vanish and the complex is called *exact* [4, p. 12]. Analogously, cochain complexes exhibit *coboundaries*, *cocycles* and *cohomology* [4, p. 16]. The terms chain, cycle and boundary might appear arbitrary but they arise naturally from one particular example, the simplicial chain complex. For details see [4, Section 2.3].

De Rham Complex

With the basic terminology established, let us look at the de Rham complex for differential forms on a smooth manifold $\Omega \subset \mathbb{R}^n$ [4, p. 16].

$$0 \longrightarrow \Lambda^0(\Omega) \xrightarrow{\mathrm{d}^0} \Lambda^1(\Omega) \xrightarrow{\mathrm{d}^1} \cdots \xrightarrow{\mathrm{d}^{n-1}} \Lambda^n(\Omega) \longrightarrow 0\,, \tag{2.26}$$

where d denotes the exterior derivative. Remember that $\mathrm{d} \circ \mathrm{d} = 0$ and hence (2.26) is indeed a complex. If we restrict ourselves to three-dimensional space, we can rewrite (2.26) in terms of vector proxies [4, p. 16]:

$$0 \longrightarrow \mathcal{C}^\infty(\Omega) \xrightarrow{\mathbf{grad}} \mathcal{C}^\infty(\Omega; \mathbb{R}^3) \xrightarrow{\mathbf{curl}} \mathcal{C}^\infty(\Omega; \mathbb{R}^3) \xrightarrow{\mathrm{div}} \mathcal{C}^\infty(\Omega) \longrightarrow 0\,. \tag{2.27}$$

In this representation it becomes apparent that the de Rham complex neatly unifies the differentiation operators **grad**, **curl** and div.

Remarkably, the de Rham complex reveals the Betti numbers, a topological invariant, of the manifold Ω. The zeroth Betti number indicates the number of connected

components, the first Betti number gives the number of tunnels through the domain and the second Betti number counts how many voids are enclosed by Ω. It is a central consequence of de Rham's theorem that the k-th Betti number equals the dimension of the k-th cohomology space of the de Rham complex [4, pp. 14 sq.]. This is one of the properties that we want to preserve when going to the discrete level.

Hilbert Complexes and Hodge Decomposition

Essential for the design and analysis of FEMs are Sobolev and Hilbert spaces. Analogously, FEEC studies Hilbert complexes to gain the pivotal structure to design finite element spaces (indeed, complexes of finite element spaces). What is more, they lead to the *Hodge decomposition* which is the central tool for developing an effective multigrid iteration for problems in $\boldsymbol{H}(\mathbf{curl})$. Thus, the Hodge decomposition will appear again in Chapter 3.

Definition 2.7 (Hilbert complex, [4, p. 33])**.** A Hilbert complex is a complex (W, d) in which the vector spaces W^k are Hilbert and the differential $d : W^k \to W^{k+1}$ is closed and densely defined. At each level there is the base space W^k and the domain space V^k of d which is also Hilbert.

The Hilbert complex which is relevant for our purposes is the L^2 de Rham complex in three dimensions which we can write using the base spaces

$$0 \longrightarrow L^2\Lambda^0(\Omega) \xrightarrow{\mathrm{d}^0} L^2\Lambda^1(\Omega) \xrightarrow{\mathrm{d}^1} L^2\Lambda^2(\Omega) \xrightarrow{\mathrm{d}^2} L^2\Lambda^3(\Omega) \longrightarrow 0 \quad (2.28)$$

or using the domain spaces [4, p. 75]

$$0 \longrightarrow H\Lambda^0(\Omega) \xrightarrow{\mathrm{d}^0} H\Lambda^1(\Omega) \xrightarrow{\mathrm{d}^1} H\Lambda^2(\Omega) \xrightarrow{\mathrm{d}^2} H\Lambda^3(\Omega) \longrightarrow 0 \ . \quad (2.29)$$

The domain spaces can alternatively incorporate homogeneous boundary conditions [13]:

$$\mathring{H}\Lambda^k(\Omega) := \left\{ \omega \in H\Lambda^k(\Omega) \mid \operatorname{tr}\omega = 0 \right\} , \quad (2.30)$$

yielding the complex [4, p. 48]

$$0 \longrightarrow \mathring{H}\Lambda^0(\Omega) \xrightarrow{\mathrm{d}^0} \mathring{H}\Lambda^1(\Omega) \xrightarrow{\mathrm{d}^1} \mathring{H}\Lambda^2(\Omega) \xrightarrow{\mathrm{d}^2} \mathring{H}\Lambda^3(\Omega) \longrightarrow 0 \, . \quad (2.31)$$

Any closed densely defined operator mapping W^k to W^{k+1} exhibits an adjoint which maps W^{k+1} to W^k [4, p. 34]. We write the adjoint of the exterior derivative d^{k-1} as d_k^*. For the complex (2.29), d_k^* has the domain space [13]

$$\mathring{H}^*\Lambda^k(\Omega) := \left\{ \omega \in L^2\Lambda^k(\Omega) \mid \mathrm{d}_k^*\omega \in L^2\Lambda^{k-1}(\Omega), \operatorname{tr}\star\omega = 0 \right\} . \quad (2.32)$$

Here, $\star$ is the Hodge star which takes k-forms to $(n-k)$-forms without fundamentally changing them. In the example of $\mathbb{R}^3$

$$\star dx^1 = dx^2 \wedge dx^3, \quad \star dx^2 = -dx^1 \wedge dx^3, \quad \star dx^3 = dx^1 \wedge dx^2 \tag{2.33}$$

and when carried over to scalar/vector proxies the Hodge star is just the identity. Details are given in [4, pp. 66, 68].

The adjoint of the exterior derivative together with its domain spaces forms itself a complex. It is known as the *dual complex* of (2.29) [13]

$$0 \longleftarrow \mathring{H}^*\Lambda^0(\Omega) \xleftarrow{\mathrm{d}_1^*} \mathring{H}^*\Lambda^1(\Omega) \xleftarrow{\mathrm{d}_2^*} \mathring{H}^*\Lambda^2(\Omega) \xleftarrow{\mathrm{d}_3^*} \mathring{H}^*\Lambda^3(\Omega) \longleftarrow 0 \ . \tag{2.34}$$

It looks a lot like the original de Rham complex, however, the spaces include boundary conditions and the indexing is reversed. The combination of both complexes and their cycle and boundary spaces leads to the Hodge decomposition of $L^2\Lambda^k(\Omega)$.

In the derivation of the Hodge decomposition below, we use $\mathring{\mathfrak{Z}}_k^*$ and $\mathring{\mathfrak{B}}_k^*$ to refer to the cycles and boundaries of the dual complex (2.34). We remark that the cycle spaces $\mathfrak{Z}^k$ and $\mathring{\mathfrak{Z}}_k^*$ are closed in $H\Lambda^k(\Omega)$ and $H^*\Lambda^k(\Omega)$, respectively. Additionally, both cycle spaces, as well as the boundary spaces $\mathfrak{B}^k$ and $\mathring{\mathfrak{B}}_k^*$ are closed in $L^2\Lambda^k(\Omega)$ [13]. This implies the following well established relations

$$\mathfrak{Z}^{k\perp} \subset \mathfrak{B}^{k\perp} = \mathring{\mathfrak{Z}}_k^* \supset \mathring{\mathfrak{B}}_k^* \,, \tag{2.35a}$$

$$\mathring{\mathfrak{Z}}_k^{*\perp} \subset \mathring{\mathfrak{B}}_k^{*\perp} = \mathfrak{Z}^k \supset \mathfrak{B}^k \,, \tag{2.35b}$$

where orthogonality is within $L^2\Lambda^k(\Omega)$ [13]. Consequently, we can orthogonally decompose $L^2\Lambda^k(\Omega)$ into $\mathfrak{B}^k$ and $\mathfrak{B}^{k\perp} = \mathring{\mathfrak{Z}}_k^*$ (2.35a). We can further decompose $\mathring{\mathfrak{Z}}_k^*$ into $\mathring{\mathfrak{B}}_k^*$ and its orthogonal complement in $\mathring{\mathfrak{Z}}_k^*$ which is (2.35b) [13]

$$\mathfrak{H}^k := \mathfrak{Z}^k \cap \mathring{\mathfrak{Z}}_k^* \,. \tag{2.36}$$

It is worth mentioning that $\mathfrak{H}^k$, the space of *harmonic k-forms*, is isomorphic to the k-th homology space $\mathcal{H}^k$ [4, p. 35]. From these results it follows directly that $L^2\Lambda^k(\Omega)$ can be orthogonally decomposed into three parts, referred to as the *Hodge decomposition* [13]

$$L^2\Lambda^k(\Omega) = \overbrace{\underbrace{\mathfrak{B}^k}_{\mathring{\mathfrak{Z}}_k^{*\perp}} \oplus \underbrace{\mathfrak{H}^k}_{\mathring{\mathfrak{Z}}_k^* = \mathfrak{B}^{k\perp}}}^{\mathfrak{Z}^k = \mathring{\mathfrak{B}}_k^{*\perp}} \oplus \overbrace{\mathring{\mathfrak{B}}_k^*}^{\mathfrak{Z}^{k\perp}} \,. \tag{2.37}$$

Let us take as example the space $L^2\Lambda^1(\Omega)$ over a domain Ω in $\mathbb{R}^3$. According to the de Rham theorem, $\dim \mathfrak{H}^1 = \dim \mathcal{H}^1$ equals the first Betti number which agrees with the number of tunnels through the domain. Consequently, if we further assume that Ω is simply connected, $\mathfrak{H}^1$ vanishes and $L^2\Lambda^1(\Omega)$ can be orthogonally decomposed into just two parts: $\mathfrak{B}^1$, the space of gradients, and $\mathring{\mathfrak{B}}_1^*$, the space of curls with homogeneous

boundary conditions. Because d and d^* are differentials, we can equivalently say that $L^2\Lambda^1(\Omega)$ is decomposed into a **curl**-free and a div-free part. Note that this is precisely the Helmholtz decomposition from vector calculus. Here, we recognize it as a special case of the Hodge decomposition [4, p. 38].

Discrete Approximation of the de Rham Complex

Now, we have all the tools needed to introduce finite element spaces related to the de Rham complex, but first, let us fix some notation. We assume that the domain $\Omega \subset \mathbb{R}^3$ is subdivided into tetrahedral elements T by a given triangulation $\mathcal{T}_h$. Additionally, we write $\mathcal{P}_r$ for the space of polynomials of degree at most r, $\mathcal{P}_r(T)$ for the same space restricted to a single tetrahedron and finally $\mathcal{P}_r(\mathcal{T}_h)$ for the space of piecewise polynomials over the triangulation. The same notation is used to denote the space of homogeneous polynomials $\mathcal{H}_r$ of degree exactly r.

Remember that we want to preserve properties of the de Rham complex (2.26) when going to the discrete level. For this to make sense, we must establish some form of this complex which consists of a finite dimensional graded vector space. In the finite element context, the canonical choice of such spaces are piecewise polynomials of fixed degree r. There are several possible choices of polynomial spaces, here we focus on $\mathcal{P}_r^-\Lambda^k \subset H\Lambda^k$ which we will define shortly. By choosing these finite spaces, we obtain the following diagram:

$$\begin{array}{ccccccc} H\Lambda^0(\Omega) & \xrightarrow{\mathrm{d}^0} & H\Lambda^1(\Omega) & \xrightarrow{\mathrm{d}^1} & H\Lambda^2(\Omega) & \xrightarrow{\mathrm{d}^2} & H\Lambda^3(\Omega) \\ \downarrow{\scriptstyle \Pi_h^0} & & \downarrow{\scriptstyle \Pi_h^1} & & \downarrow{\scriptstyle \Pi_h^2} & & \downarrow{\scriptstyle \Pi_h^3} \\ \mathcal{P}_r^-\Lambda^0(\mathcal{T}_h) & \xrightarrow{\mathrm{d}^0} & \mathcal{P}_r^-\Lambda^1(\mathcal{T}_h) & \xrightarrow{\mathrm{d}^1} & \mathcal{P}_r^-\Lambda^2(\mathcal{T}_h) & \xrightarrow{\mathrm{d}^2} & \mathcal{P}_r^-\Lambda^3(\mathcal{T}_h)\,. \end{array} \tag{2.38}$$

The first line is the Hilbert de Rham complex for differential forms on a manifold Ω in $\mathbb{R}^3$ that we have seen before. The second row is a *subcomplex* of the first. This means that for each k, $\mathcal{P}_r^-\Lambda^k$ is a subset of $H\Lambda^k$ and $\mathrm{d}^k(\mathcal{P}_r^-\Lambda^k) \subset \mathcal{P}_r^-\Lambda^{k+1}$ [4, p. 12]. The same differential d, restricted to the discrete spaces, is used in the finite de Rham complex, so like before $\mathrm{d}\circ\mathrm{d} = 0$, and the second line is indeed a complex. Note that it also follows that $\mathfrak{Z}_h \subset \mathfrak{Z}$ and $\mathfrak{B}_h \subset \mathfrak{B}$. In contrast, $\mathfrak{H}_h \nsubseteq \mathfrak{H}$ in general [4, p. 56]. Nevertheless, the continuous and discrete spaces of harmonics are isomorphic [4, Theorem 5.1] and each element from one space can be approximated well by an element of the other space [4, p. 58].

Furthermore, we require that a bounded cochain projection Π_h exists. Such a mapping has the properties that it is idempotent ($\Pi_h \circ \Pi_h = \Pi_h$), commutes with the differential ($\mathrm{d}\circ\Pi_h = \Pi_h \circ \mathrm{d}$) and is bounded in either the $H\Lambda^k(\Omega)$ or $\mathcal{P}_r^-\Lambda^k(\Omega)$ norm. It is the central result of FEEC that if the subspaces approximate the continuous spaces as $h \to 0$, the subcomplex property holds and a bounded cochain projection exists, then consistency, stability, and convergence of the mixed Galerkin method for a large class of problems, e.g. the abstract Hodge Laplacian, follow [4, p. 62].

With the criteria for an adequate discretization established, we now introduce the spaces $\mathcal{P}_r^- \Lambda^k$. To that end, we first reiterate the *complete polynomial space of differential forms*, i.e. the space of differential k-forms with polynomial coefficients

$$\mathcal{P}_r \Lambda^k := \left\{ \sum_{1 \leq \sigma_1 < \cdots < \sigma_k \leq n} p_\sigma \, dx^\sigma \mid p_\sigma \in \mathcal{P}_r \right\} \tag{2.39}$$

and define the *Koszul differential* κ of a k-form ω [4, p. 83]:

$$(\kappa\omega)_{\boldsymbol{x}}(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{k-1}) = \omega_{\boldsymbol{x}}(\underbrace{\boldsymbol{x}}_{\text{as tangent vector}}, \boldsymbol{v}_1, \ldots, \boldsymbol{v}_{k-1}),$$
$$\boldsymbol{x} \in \Omega \subset \mathbb{R}^n, \boldsymbol{v}_1, \ldots, \boldsymbol{v}_{k-1} \in \mathbb{R}^n, k \geq 1 \,. \tag{2.40}$$

For $k = 0$, $\kappa\omega := 0$. Note that in (2.40) we restrict Ω to a subset of $\mathbb{R}^n$ so that the tangent spaces $T_{\boldsymbol{x}}\Omega$ can be identified with $\mathbb{R}^n$. This allows us to reinterpret a vector in Ω as a tangent vector [4, p. 83]. For our purposes we can understand the Koszul differential as a mapping between polynomial differential forms that raises the polynomial degree and lowers the form degree [4, pp. 83 sq.]:

$$\kappa : \mathcal{P}_r \Lambda^k \to \mathcal{P}_{r+1} \Lambda^{k-1} \,. \tag{2.41}$$

In the language of vector calculus and in the case of $\mathbb{R}^3$ the Koszul differential takes the form [13]

$$k = 1 : \quad \boldsymbol{x} \mapsto \boldsymbol{x} \cdot \boldsymbol{a}(\boldsymbol{x}) \,, \tag{2.42a}$$
$$k = 2 : \quad \boldsymbol{x} \mapsto \boldsymbol{x} \times \boldsymbol{a}(\boldsymbol{x}) \,, \tag{2.42b}$$
$$k = 3 : \quad \boldsymbol{x} \mapsto \boldsymbol{x} c(\boldsymbol{x}) \,. \tag{2.42c}$$

Now we can give the definition of the *trimmed polynomial space of differential forms* [4, p. 85]

$$\mathcal{P}_r^- \Lambda^k := \mathcal{P}_{r-1} \Lambda^k + \kappa \mathcal{H}_{r-1} \Lambda^{k+1} \,. \tag{2.43}$$

From the definition we see that $\mathcal{P}_r^- \Lambda^k$ contains all polynomial forms of degree $r-1$. In addition, it contains some but not necessarily all forms of degree exactly r which arise by applying the Koszul differential to the homogeneous polynomials of degree $r-1$. It is worth pointing out that $\mathcal{P}_r^- \Lambda^0 = \mathcal{P}_r \Lambda^0$ and $\mathcal{P}_r^- \Lambda^n = \mathcal{P}_{r-1} \Lambda^n$. Conversely, for all other k, the trimmed space is strictly between the complete spaces of order $r-1$ and r: $\mathcal{P}_{r-1} \Lambda^k \subsetneq \mathcal{P}_r^- \Lambda^k \subsetneq \mathcal{P}_r \Lambda^k$ [4, p. 86]. Also note that the sum in (2.43) is direct [13]. The dimension of $\mathcal{P}_r^- \Lambda^k$ is given by [4, p. 86]

$$\dim \mathcal{P}_r^- \Lambda^k = \binom{r+n}{r+k} \binom{r+k-1}{k} \,. \tag{2.44}$$

To obtain a finite element space, we can use $\mathcal{P}_r^- \Lambda^k(T)$ as the space of shape functions on each element $T \in \mathcal{T}_h$. What is left to define is a set of degrees of freedom (DoFs).

For each d-dimensional subsimplex f of an element T they are given by [4, p. 87]

$$\omega \mapsto \int_f (\operatorname{tr}_f \omega) \wedge \mu, \quad \mu \in \mathcal{P}_{r+k-d-1}\Lambda^{d-k}(f), f \in \triangle_d(T), d \geq k\,. \tag{2.45}$$

It is possible to show that this set of DoFs is unisolvent and consequently, together with the triangulation and shape functions it defines a finite element space. A nice property of these DoFs is that they enforce exactly the continuity which is required for a finite element function to be in $H\Lambda^k$ [4, p. 88]. In the case of $\boldsymbol{H}(\mathbf{curl})$ ($k = 1$), this means that only tangential components are continuous [7]. Furthermore, the projection operators defined by (2.45) commute with the exterior derivative and hence form a cochain projection [4, p. 88]. However, they are not bounded. Hence, for much of the theoretical consideration it must be resorted to alternative projection operators. In fact, the construction of projections which fulfill certain desired properties is still an active area of research [4, p. 90].

For the implementation we will make heavy use of the fact that the spaces $\mathcal{P}_r^-\Lambda^k$ are affine invariant. This means that instead of defining the element local space $\mathcal{P}_r^-\Lambda^k(T)$ as the restriction of $\mathcal{P}_r^-\Lambda^k$ to a tetrahedron $T \in \mathcal{T}_h$, it can be equivalently viewed as the pullback under an affine transformation from a reference tetrahedron $\hat{T}$ [13]. So if $F : \hat{T} \to T$ is a diffeomorphic mapping from the reference element to the affine simplex T, we can use the pullback of the inverse mapping to obtain the affine space [15]

$$\mathcal{P}_r^-\Lambda^k(T) = \left(F^{-1}\right)^* \left(\mathcal{P}_r^-\Lambda^k(\hat{T})\right)\,. \tag{2.46}$$

This construction makes it possible to perform all computations, e.g. integrals over elements, on the reference tetrahedron.

The spaces $\mathcal{P}_r^-\Lambda^k$ appeared in the finite element literature before their systematic analysis through FEEC. The Lagrange finite element space of continuous, piecewise polynomials of degree r is the same as $\mathcal{P}_r^-\Lambda^0$. $\mathcal{P}_r^-\Lambda^1$ and $\mathcal{P}_r^-\Lambda^2$ are the Nédélec spaces of type I for $\boldsymbol{H}(\mathbf{curl})$ and $\boldsymbol{H}(\mathrm{div})$, respectively. Finally, $\mathcal{P}_{r+1}^-\Lambda^3$ is the space of discontinuous piecewise polynomials of degree r, which is used in the discontinuous Galerkin method [4, pp. 93 sq.].

The Nédélec Space of Type I and Order 1

Since the model problem searches a solution $\boldsymbol{u} \in \boldsymbol{H}(\mathbf{curl})$, the space relevant for our problem is the Nédélec space of edge elements, i.e. the case $k = 1$. It has been first introduced by Nédélec [7]. Moreover, we choose linear polynomials and set $r = 1$. Inserting k and r into (2.43), we find the space of shape functions, expressed here in terms of vector proxies:

$$\mathcal{P}_1^-\Lambda^1 = \left\{ \boldsymbol{x} \mapsto \boldsymbol{a} + \boldsymbol{b} \times \boldsymbol{x} \mid \boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^3 \right\}\,. \tag{2.47}$$

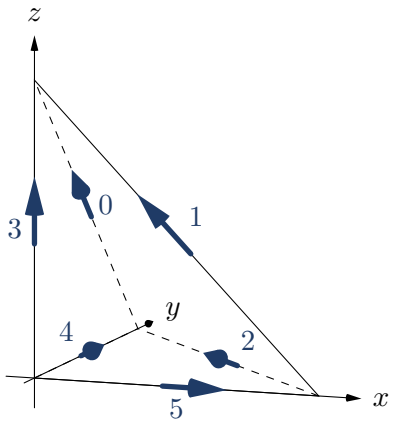


Figure 2.2: DoFs of a tetrahedral Nédélec element of Type I, order 1.

In the same way the DoFs (2.45) simplify to

$$\boldsymbol{u} \mapsto \int_e \boldsymbol{u}(\boldsymbol{x}) \cdot \boldsymbol{t} \, \mathrm{d}\boldsymbol{x}, \quad e \in \triangle_1(\mathcal{T}_h)\,, \tag{2.48}$$

where $\boldsymbol{t}$ is the unit-length tangent vector of the edge e. From (2.48) it becomes evident that there is one DoF associated to each edge of the mesh $\mathcal{T}_h$.

Lastly, we derive the basis functions linked to this set of DoFs. We start off with the reference tetrahedron

$$\hat{T} = \left\{ \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \mid x_1, x_2, x_3 \geq 0, x_1 + x_2 + x_3 \leq 1 \right\}. \tag{2.49}$$

$\mathcal{P}_1^- \Lambda^1$ (2.47) is canonically spanned by the columns of the matrix

$$\mathcal{B}_{\text{canonical}} = \begin{pmatrix} 1 & 0 & 0 & 0 & z & -y \\ 0 & 1 & 0 & -z & 0 & x \\ 0 & 0 & 1 & y & -x & 0 \end{pmatrix}. \tag{2.50}$$

From this we can determine the finite element basis functions associated to each DoF. To that end, we calculate the DoFs for each canonical basis function and collect them in the matrix D, where rows correspond to edges and columns to basis functions. Indexing and orientation of the edges are indicated in Figure 2.2. On the reference

element the basis then reads

$$\begin{aligned}\mathcal{B}_{\mathrm{FEM}}(\hat{T}) = \mathcal{B}_{\mathrm{canonical}} D^{-1} = \mathcal{B}_{\mathrm{canonical}} \begin{pmatrix} 0 & -1 & 1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 & -1 & 0 \\ -1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}^{-1} \\ = \begin{pmatrix} 0 & -z & -y & z & y & -y-z+1 \\ -z & 0 & x & z & -x-z+1 & x \\ y & x & 0 & -x-y+1 & y & x \end{pmatrix}.\end{aligned} \tag{2.51}$$

When given the DoFs $\boldsymbol{s} \in \mathbb{R}^6$ of a Nédélec function $\boldsymbol{u}$ on $\hat{T}$, this function can be evaluated at a point $\boldsymbol{x} \in \hat{T}$ by

$$\boldsymbol{u}(\boldsymbol{x}) = \mathcal{B}_{\mathrm{FEM}}(\boldsymbol{x})\boldsymbol{s}\,. \tag{2.52}$$

To obtain the basis on an affine tetrahedron, we use the pullback. Assume that an affine tetrahedron T is defined by the affine map

$$F : \hat{T} \to T, \quad \hat{\boldsymbol{x}} \mapsto B\hat{\boldsymbol{x}} + \boldsymbol{b}\,. \tag{2.53}$$

Using (2.46), a basis function $\hat{\boldsymbol{\phi}}$ on $\hat{T}$ transforms to the basis function $\boldsymbol{\phi}$ on T according to

$$\hat{\boldsymbol{\phi}} \mapsto \left(F^{-1}\right)^* \left(\hat{\boldsymbol{\phi}} \circ F^{-1}\right) = B^{-T} \left(\hat{\boldsymbol{\phi}} \circ F^{-1}\right). \tag{2.54}$$

Note that (2.54) is the covariant Piola transform [16].

While basis functions in Euclidean coordinates are very useful for the implementation, there is a more compact representation using barycentric coordinates. Let λ_i be the barycentric coordinate connected to vertex i of a tetrahedron T. A basis of $\mathcal{P}_1^- \Lambda^1(T)$ is then given by [17]

$$\{\, \lambda_i \mathrm{d}\lambda_j - \lambda_j \mathrm{d}\lambda_i, \forall e_{ij} \in \triangle_1(T) \,\}\,. \tag{2.55}$$

Here, e_{ij} is the directed edge from vertex i to j. (2.55) is translated to vector proxies by replacing d with **grad** [17]. We will use the barycentric basis to derive the prolongation operator in Section 4.3.1.

CHAPTER

3 Multigrid

3.1 General Multigrid Theory

Discretizing a partial differential equation (PDE) leads to a linear system of equations (LSE) with typically $n = \Theta(h^{-d})$ unknowns, with the Landau symbol Θ, grid spacing h and dimension d. Often h must be chosen small enough to achieve a certain accuracy and hence the system can become quite large. Therefore, it is imperative to use a scalable solver. Ideally, the computational work to solve a system up to discretization error does not grow faster than the number of unknowns. In an iterative scheme like multigrid this is attained by meeting two conditions: requiring a number of iterations which is independent of the mesh width and having optimal complexity for a single iteration. Few algorithms achieve this remarkable property. Under certain circumstances, (full) multigrid is one of them [3, p. 20].

3.1.1 Multigrid for the Poisson Problem

Multigrid is most naturally introduced as a solver for the Poisson problem (in one dimension) [18, p. 1]

$$-u'' = f \quad x \in (0,1)\,, \tag{3.1a}$$

$$u = 0 \quad x \in \{0,1\}\,. \tag{3.1b}$$

When discretizing (3.1) on an equidistant grid with n subintervals and mesh size h by second order central finite differences, we obtain an LSE of the form [18, p. 1]

$$A\boldsymbol{u} := \begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix} \begin{pmatrix} u_1 \\ \vdots \\ u_{n-1} \end{pmatrix} = h^2 \begin{pmatrix} f_1 \\ \vdots \\ f_{n-1} \end{pmatrix} =: \boldsymbol{f}\,. \tag{3.2}$$

The spectrum of A is important for the analysis below. The eigenvalues of A are

$$\lambda_k(A) = 4\sin^2\left(\frac{k\pi}{2n}\right), \quad 1 \le k \le n-1\,, \tag{3.3}$$

and the j-th component of the corresponding eigenvector is given by [18, p. 18]

$$w_{k,j}(A) = \sin\left(\frac{jk\pi}{n}\right), \quad 1 \leq k \leq n-1, \quad 0 \leq j \leq n\,. \tag{3.4}$$

Remarkably, the eigenvectors (3.4) are nothing else but the Fourier modes which is why we refer to k as the *wavenumber* [18, p. 12]. While small wavenumbers correspond to smooth eigenvectors, large wavenumbers describe the oscillatory part of the spectrum. Formally, we say a wave in the lower half of the spectrum, i.e. $k < \frac{n}{2}$, is smooth and a mode in the upper half of the spectrum ($k \geq \frac{n}{2}$) is oscillatory [18, p. 19].

Moreover, the eigenvalues with small k are close to 0 whereas $\lambda_k(A)$ gets close to 1 as k is increased. Consequently, the larger the eigenvalue, the more oscillating the accompanying eigenvector is. This is a property which elliptic operators have in common and it is crucial for the efficacy of multigrid [19, 20].

One way to solve (3.2) is by means of a classical stationary iterative scheme like the Jacobi or Gauss-Seidel method. To that end, we split the system matrix A in its diagonal and strictly lower and upper diagonal parts D, $-L$ and $-U$, respectively, as $A = D - L - U$. The Jacobi method can then be written as

$$\boldsymbol{u}^{(m+1)} = D^{-1}(L+U)\boldsymbol{u}^{(m)} + D^{-1}\boldsymbol{f}\,, \tag{3.5}$$

and the more general weighted Jacobi method takes the form

$$\boldsymbol{u}^{(m+1)} = \Big((1-\omega)I + \omega D^{-1}(L+U)\Big)\,\boldsymbol{u}^{(m)} + \omega D^{-1}\boldsymbol{f} \tag{3.6a}$$

$$=: R_\omega \boldsymbol{u}^{(m)} + \omega D^{-1}\boldsymbol{f}\,. \tag{3.6b}$$

We refer to R_ω as the weighted Jacobi iteration matrix [18, p. 9]. Similarly, the Gauss-Seidel method is given by [18, p. 11]

$$\boldsymbol{u}^{(m+1)} = (D-L)^{-1}U\boldsymbol{u}^{(m)} + (D-L)^{-1}\boldsymbol{f}\,. \tag{3.7}$$

In the following, we focus on the weighted Jacobi method.

The weighted Jacobi and Gauss-Seidel iterations have a trait called smoothing property. While this makes them unsuitable as a standalone solver for large systems, it forms the base of the power of the multigrid algorithm. To understand this property, we look at the homogeneous system $A\boldsymbol{u} = \boldsymbol{0}$ because in this case we know the error at iteration m

$$\boldsymbol{e}^{(m)} = \boldsymbol{u}^* - \boldsymbol{u}^{(m)} \tag{3.8}$$

exactly. Since the analytical solution $\boldsymbol{u}^*$ is zero, the error is given by $\boldsymbol{e}^{(m)} = -\boldsymbol{u}^{(m)}$. It can be shown that the error after m iterations follows the relationship $\boldsymbol{e}^{(m)} = R_\omega^m \boldsymbol{e}^{(0)}$. We can expand this expression in terms of the eigenvectors of A, which coincide with the eigenvectors of R_ω [18, p. 19]:

$$\boldsymbol{e}^{(m)} = R_\omega^m \boldsymbol{e}^{(0)} = \sum_{k=1}^{n-1} c_k R_\omega^m \boldsymbol{w}_k = \sum_{k=1}^{n-1} c_k \lambda_k^m(R_\omega)\boldsymbol{w}_k\,, \tag{3.9}$$

where the eigenvalues of R_ω relate to the eigenvalues of A by [18, pp. 17 sq.]

$$\lambda_k(R_\omega) = 1 - \frac{\omega}{2}\lambda_k(A) = 1 - 2\omega \sin^2\left(\frac{k\pi}{2n}\right), \quad 1 \leq k \leq n-1\,. \tag{3.10}$$

From (3.9) we learn that the eigenvalues of the iteration matrix determine how fast error components with respect to the eigenvectors of A are damped. Note that $\lambda_k(R_\omega)$ is close to 1 for small k and decreases for larger k (depending on ω). Consequently, provided ω is chosen adequately, oscillatory error components are damped much more quickly than smooth components. Phrased differently, while the oscillatory error components vanish after a few Jacobi iterations, smooth modes require many iterations to get rid of. That is what is called the smoothing property of the Jacobi method.

By choosing ω appropriately we can accelerate smoothing of the high-frequency modes. For the Poisson problem, $\omega = \frac{2}{3}$ is the optimal choice as it ensures that $|\lambda_k| < \frac{1}{3}$ for all oscillatory k, independent of h. Conversely, no choice of ω can guarantee such a bound for the smooth modes. Instead, λ_1 is always close to 1 for any convergent value of ω [18, p. 21].

Another important observation is that the distinction in smooth and oscillatory components depends on the grid spacing h. In particular, a mode that is smooth on a fine grid appears more oscillatory on a coarser grid [18, p. 31]. This lays the foundation of the central idea in multigrid: to handle different error components on different grids. More precisely, oscillating error components are removed from the initial guess by performing a couple steps of a relaxation scheme. The remaining, smooth modes of the error are then determined on a coarser grid, where they appear more oscillatory. Finally, the fine grid approximation is corrected accordingly [18, p. 33].

This raises the question of how the error on the coarse grid can be found. It is easy to see that the error solves the residual equation [18, p. 8]

$$A\boldsymbol{e} = \boldsymbol{r}\,. \tag{3.11}$$

This relationship follows directly from the definitions of the error (3.8) and the residual

$$\boldsymbol{r} = \boldsymbol{f} - A\boldsymbol{u}\,. \tag{3.12}$$

Since the residual can be determined with little effort, finding the error contained in a given iterate is essentially the same problem as solving the original system. It can thus be tackled recursively with the same techniques. To reiterate, the smooth error components are calculated on a coarser grid by solving (3.11) and then the current approximation $\boldsymbol{u}$ is corrected to arrive at the exact solution $\boldsymbol{u}^*$:

$$\boldsymbol{u}^* = \boldsymbol{u} + \boldsymbol{e}\,. \tag{3.13}$$

Despite written as an exact equality here, in practice only an approximation is obtained hence the process must be repeated until convergence.

So far we assumed that grids with different mesh widths h are readily available. In practice, it is common to mesh the domain with a coarse grid first. The finer grids are then obtained from the coarsest grid by successive refinement. This yields a hierarchy of nested grids where the grid spacing reduces by factor 2 when going up one level.

Lastly, we need a way to transfer vectors between grids. We denote the prolongation operator, which transfers coarse grid vectors with grid spacing $2h$ to a fine grid with grid spacing h, by I_{2h}^{h}. In conforming finite element methods (FEMs) it arises naturally from the embedding of the coarse grid function space in the fine grid function space. The restriction operator, written I_{h}^{2h} does the reverse transfer and is defined as the transpose of the prolongation [21, p. 231]. Furthermore, the system matrix A^h can be found on each grid by discretizing with the particular grid spacing h.

Now we have all elements that are needed to write down the V-cycle multigrid scheme in Algorithm 3.1 [18, pp. 40 sq.][22]. First, on the fine level the oscillating error components are removed by relaxation (Line 4). Next, the remaining modes are determined recursively by solving (3.11) in Lines 5 to 7. Subsequently, the error is prolongated to the fine grid and the current iterate $\boldsymbol{u}^h$ is corrected (Line 8). Finally, in Line 9, it is relaxed on the fine grid again. Once a sufficiently coarse grid is reached, the problem is solved exactly, e.g. with a direct or Krylov subspace solver (Line 2).

Algorithm 3.1: Multigrid V(ν_1, ν_2)-cycle.

Input: initial guess $\boldsymbol{u}^h$, right-hand side (RHS) $\boldsymbol{f}^h$
Output: approximate solution $\boldsymbol{u}^h$

```
1  if Ω^h is coarsest grid then
2  |  Solve A^h u^h = f^h exactly                          /* coarse-grid solve */
3  else
4  |  Relax ν_1 times on A^h u^h = f^h with given initial guess  /* pre-smoothing */
5  |  r^{2h} ← I_h^{2h}(f^h − A^h u^h)                     /* restriction of residual */
6  |  e^{2h} ← 0
7  |  e^{2h} ← V-cycle(e^{2h}, r^{2h})                    /* solve recursively on coarse grid */
8  |  u^h ← u^h + I_{2h}^h e^{2h}                          /* prolongation and correction */
9  |  Relax ν_2 times on A^h u^h = f^h                     /* post-smoothing */
10 end
```

3.1.2 Chebyshev Smoother

The smoothers introduced so far are the Jacobi and Gauss-Seidel relaxations. However, both schemes have their drawbacks. The performance of Jacobi's method hinges on an appropriate choice for the weight ω. In practice, the optimal weight is not always easy to find. Gauss-Seidel relaxation is sequential by nature and therefore not

suited for highly parallel computing. Admitting that coloring techniques make Gauss-Seidel parallel, they are potentially hard to implement and require a comparably large amount of communication [23]. An idea to overcome these shortcomings are (weighted) blockwise smoothers which operate on a partition of the domain. While unknowns inside the blocks are updated by a Gauss-Seidel smoother, the blocks themselves are combined using Jacobi relaxation. Although this strategy can work well in practice, its performance reportedly deteriorates as the number of processors is increased [23].

Another approach are polynomial smoothers which achieve good smoothing properties independent of the number of processors as long as the polynomial degree is sufficiently high [24]. They have the additional benefit of requiring only an implementation of the matrix-vector multiplication. Given that this routine must be implemented, optimized and parallelized anyway, overall development time can be reduced. Moreover, the performance of the smoother is independent of the order of unknowns and their distribution to processes. On the downside, this type of smoother requires eigenvalue estimates of the system matrix [23].

Our description of the Chebyshev smoother follows the presentation in [24]. Let $A'\boldsymbol{u} = \boldsymbol{f}'$ be the system to be solved. To keep the notation compact, the system is scaled by the inverse diagonal D^{-1} of A' so that the diagonal reduces to unity: $A := D^{-1}A'$, $\boldsymbol{f} := D^{-1}\boldsymbol{f}'$. The scaled system $A\boldsymbol{u} = \boldsymbol{f}$ can then be solved by the iterative procedure

$$\boldsymbol{u}^{(m+1)} = \boldsymbol{u}^{(m)} + p_r(A)\left(\boldsymbol{f} - A\boldsymbol{u}^{(m)}\right), \tag{3.14}$$

where p_r is a polynomial of degree r:

$$p_r(A) = \sum_{i=0}^{r} \alpha_i A^i . \tag{3.15}$$

It can be shown that the error of the iteration (3.14) is damped by the polynomial q_{r+1} according to

$$\boldsymbol{e}^{(m+1)} = q_{r+1}(A)\boldsymbol{e}^{(m)} := (I - p_r(A)A)\boldsymbol{e}^{(m)} . \tag{3.16}$$

Note that (3.16) implies that $q_{r+1}(0) = 1$. Thus, we introduce $\mathcal{P}_r^1$, the set of polynomials of degree r which evaluate to 1 at $x = 0$.

As it is our goal to develop a smoothing operator, the scheme (3.14) shall dampen the upper half of the spectrum of A. This is achieved by minimizing the maximum absolute value that q_{r+1} attains in the interval $[\lambda_{n_c+1}, \lambda_n]$, where λ_{n_c+1} is the smallest eigenvalue of A that can not be handled on the coarse grid and λ_n is the largest eigenvalue of A. Formally, we seek the solution to the minimization problem

$$q_{r+1} = \operatorname*{arg\,min}_{p_{r+1} \in \mathcal{P}_{r+1}^1} \max_{x \in [\lambda_{n_c+1}, \lambda_n]} |p_{r+1}(x)| . \tag{3.17}$$

Fortunately, q_{r+1} can be found with simple recurrence relations involving the two eigenvalues λ_{n_c+1} and λ_n. This is where Chebyshev polynomials come into play. They are defined as [25, p. 639]

$$T_r(x) = \frac{1}{2}\left(\left(x + \sqrt{x^2-1}\right)^r + \left(x - \sqrt{x^2-1}\right)^r\right), \quad r = 0, 1, \ldots \tag{3.18}$$

and give the solution to the minimization problem (3.17) over the interval $x \in [a, b]$:

$$q_{r+1} = \frac{T_{r+1}\left(\frac{b+a-2x}{b-a}\right)}{T_{r+1}\left(\frac{b+a}{b-a}\right)}. \tag{3.19}$$

A proof of this property is given in [25, p. 640]. Moreover, $|q_{r+1}(x)| < 1$ for all $x \in (0, b]$ which implies that convergence of the Chebyshev smoother is guaranteed if the spectrum of A is contained in $(0, b]$ [23].

It remains to find estimates of the eigenvalues λ_{n_c+1} and λ_n. The extremal eigenvalue λ_n can easily be estimated with a few iterations of conjugate gradient (CG) or power iteration [23]. While it is important not to underestimate λ_n, multigrid convergence is not very sensitive to the estimate of the lower bound. Therefore, it is a valid strategy to simply scale the upper eigenvalue bound by a constant factor < 1 to estimate λ_{n_c+1} [24].

3.2 Multigrid in $\boldsymbol{H}(\mathbf{curl})$

So far we have considered the Poisson problem and have learned that its spectral properties are fundamental to the smoothing property of stationary iterations. Can we apply the same techniques to our problem (2.2)? Unfortunately, frequently used multigrid smoothers like Gauss-Seidel or Chebyshev do not possess the smoothing property when applied to (2.2) [19]. Here we want to verify this by performing a numerical experiment.

3.2.1 Numerical Experiment with Chebyshev Smoother

For our experiment we take as domain the unit cube $\Omega = (0, 1)^3$ and try solving (2.2) with $\alpha = \beta = 1$. Ω is subdivided into six macro-tetrahedra which in turn are further uniformly refined four times. The resulting mesh has a total of 31 024 degrees of freedom (DoFs). For more details on hybrid tetrahedral grids refer to Chapter 4. In terms of discretization, we rely on linear Nédélec edge elements of the first kind. Furthermore, the RHS is set to 0 so that we know the error in each iteration exactly. We solve the system by performing up to 100 Chebyshev smoothing iterations and count the number of iterations needed to reduce the initial error, measured in the 2-norm, by factor 5. As initial guess for $\boldsymbol{u}$ we try three different oscillating functions where in each case increasing k makes the function more oscillatory:

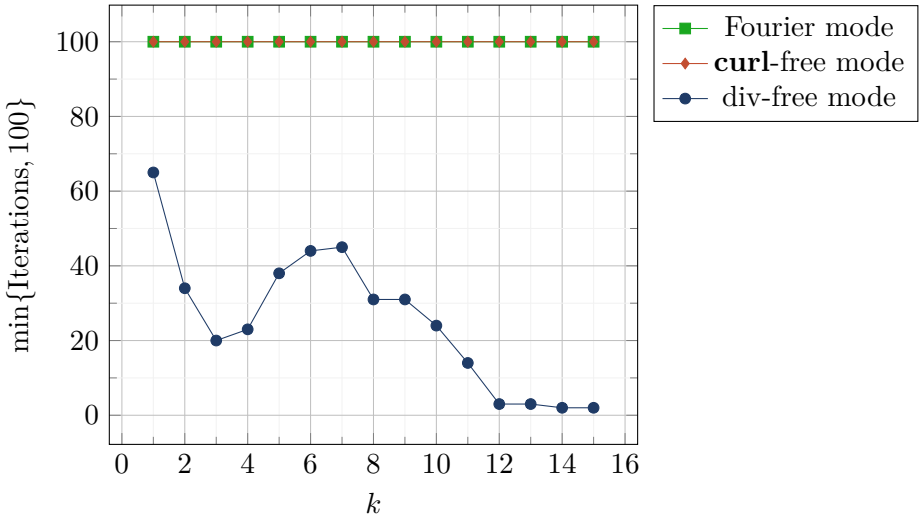


Figure 3.1: Number of Chebyshev smoothing iterations to reduce the initial error $\|\boldsymbol{e}\|_2$ by factor 5.

Fourier mode Each component of $\boldsymbol{u}_\mathrm{f}$ is the classical Fourier mode.

$$\boldsymbol{u}_\mathrm{f} = \begin{pmatrix} \sin(k\pi x)\sin(k\pi y)\sin(k\pi z) \\ \sin(k\pi x)\sin(k\pi y)\sin(k\pi z) \\ \sin(k\pi x)\sin(k\pi y)\sin(k\pi z) \end{pmatrix} \tag{3.20}$$

curl-free mode This function has the same frequency as (3.20) but $\boldsymbol{u}_\mathrm{cf} \in \mathcal{N}(\mathbf{curl})$.

$$\boldsymbol{u}_\mathrm{cf} = \begin{pmatrix} \cos(k\pi x)\sin(k\pi y)\sin(k\pi z) \\ \sin(k\pi x)\cos(k\pi y)\sin(k\pi z) \\ \sin(k\pi x)\sin(k\pi y)\cos(k\pi z) \end{pmatrix} \tag{3.21}$$

div-**free mode** This function has the same frequency as (3.20) but $\boldsymbol{u}_\mathrm{df} \in \mathcal{N}(\mathbf{curl})^\perp$.

$$\boldsymbol{u}_\mathrm{df} = \begin{pmatrix} \sin(k\pi x)\cos(k\pi y)\sin(k\pi z) - \sin(k\pi x)\sin(k\pi y)\cos(k\pi z) \\ \sin(k\pi x)\sin(k\pi y)\cos(k\pi z) - \cos(k\pi x)\sin(k\pi y)\sin(k\pi z) \\ \cos(k\pi x)\sin(k\pi y)\sin(k\pi z) - \sin(k\pi x)\cos(k\pi y)\sin(k\pi z) \end{pmatrix} \tag{3.22}$$

The results are summarized in Figure 3.1. Unfortunately, the Chebyshev smoother is not able to reduce the error in general. Instead, only in the div-free case we can see a reduction and even there it is suboptimal. As expected, convergence is fastest for highly oscillatory modes. However, the curve defies expectations by exhibiting a prominent peak in the center of the spectrum. What is more, although not visible in Figure 3.1, even for large k the error is not reduced significantly further than factor 5.

3.2.2 Helmholtz Decomposition

The reason why the Chebyshev smoother failed completely for the Fourier and **curl**-free modes can be understood with the help of the Hodge decomposition (2.37). Remember that the smoothing property hinges on ellipticity of the system matrix A. However, it is easy to see that the bilinear form of the model problem

$$a(\boldsymbol{u}, \boldsymbol{v}) = \alpha(\mathbf{curl}\, \boldsymbol{u}, \mathbf{curl}\, \boldsymbol{v})_{\boldsymbol{L}^2(\Omega)} + \beta(\boldsymbol{u}, \boldsymbol{v})_{\boldsymbol{L}^2(\Omega)}, \quad \alpha, \beta > 0 \tag{3.23}$$

reduces to the identity on $\mathcal{N}(\mathbf{curl})$. The identity clearly is not elliptic since the eigenvalues do not depend on the frequency of the eigenvectors. On the other hand, $a(\cdot, \cdot)$ behaves like a second order elliptic operator when applied to functions in $\mathcal{N}(\mathbf{curl})^\perp$ [19]. This can be shown by looking at Fourier transformations of weakly divergence-free functions in $\mathring{\boldsymbol{H}}(\mathbf{curl}, \mathbb{R}^3)$ [20]:

$$(\mathbf{curl}\, \boldsymbol{w}, \mathbf{curl}\, \boldsymbol{w}) = \int_{\mathbb{R}^3} |\boldsymbol{k}|^2 |\hat{\boldsymbol{w}}(\boldsymbol{k})|^2 \, \mathrm{d}\boldsymbol{k}, \quad (\boldsymbol{w}, \mathbf{grad}\, \phi) = 0 \quad \forall \phi \in H^1(\mathbb{R}^3)\,, \tag{3.24}$$

where $\hat{\boldsymbol{w}}$ is the Fourier transformation of $\boldsymbol{w}$ and $\boldsymbol{k}$ stands for the wave vector in Fourier space. It can be seen from (3.24) that functions in $\mathcal{N}(\mathbf{curl})^\perp$ are amplified by the square of their frequency under the action of $a(\cdot, \cdot)$. This essentially shows the elliptic character of the bilinear form (3.23) [20]. Note that even with the ellipticity in $\mathcal{N}(\mathbf{curl})^\perp$ of $a(\cdot, \cdot)$ established, stationary iterations are only effective if the nodal basis functions are sufficiently orthogonal to $\mathcal{N}(\mathbf{curl})$. This is the case for Nédélec edge elements and another reason for the choice of this finite element space [8, 20].

This motivates that $\mathcal{N}(\mathbf{curl})^\perp$ can be satisfactorily handled by a standard Chebyshev smoother. While in our numerical experiment in Section 3.2.1 we observed mixed results for the function in $\mathcal{N}(\mathbf{curl})^\perp$, there are reasons why $\boldsymbol{u}_{\mathrm{df}}$ is not precisely div-free in the discrete sense. Most notably, we approximate the integral in the projection (2.48) by the midpoint rule. Let us hence assume that $\mathcal{N}(\mathbf{curl})^\perp$ can be satisfactorily handled by the Chebyshev smoother. It is left to find an effective means to handle the remaining components which lie in $\mathcal{N}(\mathbf{curl})$.

To make the analysis simpler, we restrict ourselves to homogeneous boundary conditions and cases in which Ω is a convex bounded subset of $\mathbb{R}^3$. In Section 5.7 we show that the algorithm is applicable to a wider range of cases than shown here. These assumptions ensure that the de Rham complex (2.31) is exact. Therefore, we can deduce directly that **curl**-free functions can be represented by gradients of scalar potentials:

$$\mathring{\boldsymbol{H}}(\mathbf{curl}; \Omega) \cap \mathcal{N}(\mathbf{curl}) = \mathbf{grad}\, \mathring{H}^1(\Omega)\,. \tag{3.25}$$

Owing to the structure preserving discretization, the same is true on the discrete level [8], cf. (2.38):

$$\mathring{\mathcal{P}}_r^- \Lambda^1(\mathcal{T}_h) \cap \mathcal{N}(\mathbf{curl}) = \mathbf{grad}\, \mathring{\mathcal{P}}_r^- \Lambda^0(\mathcal{T}_h)\,. \tag{3.26}$$

It is a unique and pivotal feature of the Nédélec finite element space for $\boldsymbol{H}(\mathbf{curl})$ that the potential space on the discrete level is itself a finite element space, meaning for instance that a local basis exists [8].

These representations can also be explained by looking at the Hodge decomposition (2.37). Since we posed the condition that Ω is simply connected, the harmonics vanish and $\mathcal{N}(\mathbf{curl}) = \mathfrak{Z}^1 = \mathfrak{B}^1$, which is the space of gradients. If we were to relax the condition on Ω, the harmonic forms would not be captured by the representation as a gradient and thus not handled by the smoother. Note, however, that the dimension of the space of harmonics is usually low and that harmonic forms are smooth. It follows that the coarse grid solver takes care of any harmonics and the multigrid algorithm remains applicable even if Ω is more complex [8].

The representations (3.25) and (3.26) allow us to rewrite $a(\boldsymbol{u}, \boldsymbol{v})$ for $\boldsymbol{u}, \boldsymbol{v} \in \mathcal{N}(\mathbf{curl})$ as

$$a(\boldsymbol{u}, \boldsymbol{v}) = \beta(\mathbf{grad}\,\phi, \mathbf{grad}\,\psi)_{\boldsymbol{L}^2(\Omega)}, \quad \beta > 0, \quad \phi, \psi \in \mathring{H}^1(\Omega)\,, \tag{3.27}$$

which is the weak form of the Laplace operator. This suggests that we can use standard multigrid smoothers to handle $\mathcal{N}(\mathbf{curl})$.

3.2.3 Hybrid Smoother

The discussion above motivates the hybrid smoother for problems in $\boldsymbol{H}(\mathbf{curl})$, which is given in Algorithm 3.2. For a rigorous analysis the reader is referred to [8]. The hybrid smoother performs in each iteration two smoothing steps with one of the traditional relaxation schemes. First, error components which fall in $\mathcal{N}(\mathbf{curl})^\perp$ are damped by relaxing on $A\boldsymbol{u} = \boldsymbol{f}$ directly in $\mathcal{P}_1^- \Lambda^1$ (Line 1). The remaining modes, which are in the nullspace of the **curl**-operator, are handled by lifting the vector-valued residual $\boldsymbol{r}$ to potential space (Lines 2 to 3). The lifting operator effects the opposite of the gradient, hence it is defined as the transpose of the gradient. Then, in Line 5, error components in potential space are determined by relaxing on the residual equation

$$\beta\,\Delta e = r\,, \tag{3.28}$$

where Δ is the scalar Laplace operator, r is the scalar-valued residual and e the error in potential space. Both e and r are elements of $\mathcal{P}_1^- \Lambda^0$. Lastly, e is transformed back to the Nédélec space by taking the gradient and added to the current iterate to correct for the error e (Line 6). The implementation of the grid transfer operators is detailed in the next chapter.

3.2.4 Numerical Experiment with Hybrid Smoother

After seeing the Chebyshev smoother fail in Section 3.2.1 we repeat the experiment, this time using the hybrid smoother. For relaxation in $\mathcal{P}_1^- \Lambda^1$ we use the Chebyshev

Algorithm 3.2: Hybrid smoother.

Input: initial guess $\boldsymbol{u}^h \in \mathcal{P}_1^- \Lambda^1$, RHS $\boldsymbol{f}^h \in \mathcal{P}_1^- \Lambda^1$
Output: smoothed iterate $\boldsymbol{u}^h$

1 Relax on $A^h \boldsymbol{u}^h = \boldsymbol{f}^h$ `/* Smooth on` $\mathcal{N}(\mathbf{curl})^\perp$ `*/`
2 $\boldsymbol{r}^h \leftarrow \boldsymbol{f}^h - A^h \boldsymbol{u}^h$ `/* Calculate residual */`
3 $r^h \leftarrow L^h \boldsymbol{r}^h$ `/* Lift residual to potential space */`
4 $e^h \leftarrow 0$
5 Relax on $\beta \, \Delta^h e^h = r^h$ `/* Smooth on` $\mathcal{N}(\mathbf{curl})$ `*/`
6 **return** $\boldsymbol{u}^h + L^{*h} e^h$ `/* Correct for error in` $\mathcal{N}(\mathbf{curl})$ `*/`

smoother as before but now we additionally smooth in potential space with the Gauss-Seidel method. The setup is the same as before barring now we count the iterations until the initial error is reduced by factor 100.

The results are depicted in Figure 3.2. It is clear that the hybrid smoother handles all three functions well. For modes in the upper half of the spectrum less than ten iterations are required to reduce the initial error by factor 100. In contrast to the Chebyshev smoother, the hybrid smoother does not stall after a moderate error reduction.

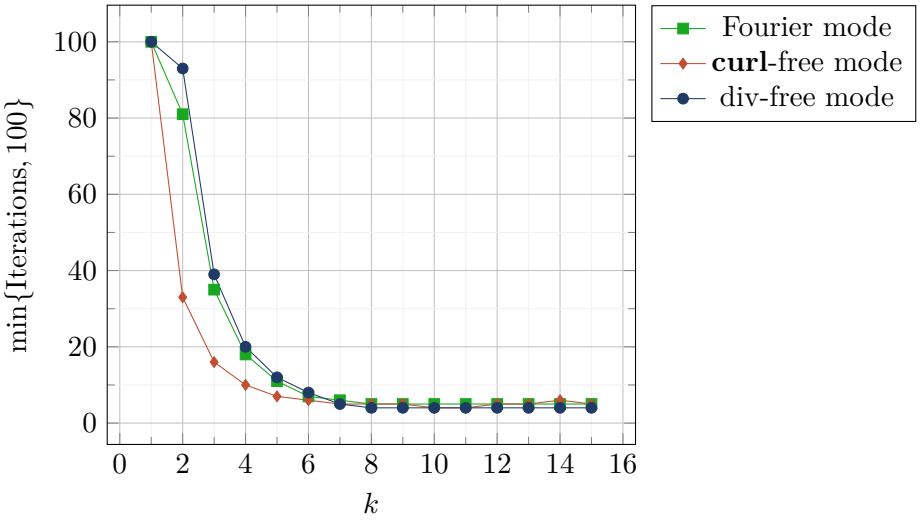


Figure 3.2: Number of hybrid smoothing iterations to reduce the initial error $\|\boldsymbol{e}\|_2$ by factor 100.

Chapter 4

Implementation

This chapter presents our implementation of the Nédélec finite element space and the hybrid smoother (Algorithm 3.2) in the *HyTeG*[1] finite element framework [9].

4.1 The *HyTeG* Finite Element Framework

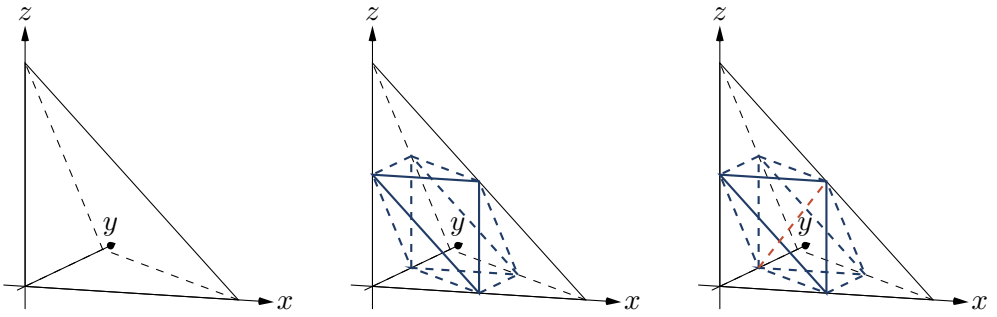
HyTeG is a finite element framework designed for massively parallel compute architectures [26]. It supersedes the HHG framework which was already capable of solving systems with $1.1 \cdot 10^{13}$ unknowns [27]. The key building block to achieve these impressive results is a matrix-free implementation of geometric multigrid on hybrid tetrahedral grids. In *HyTeG* multigrid can be used as a standalone solver or preconditioner of a Krylov subspace iteration. Various Krylov solvers are implemented internally and an interface to external libraries like PETSc [28] is offered. *HyTeG* is implemented in C++ and uses hybrid parallelization with OpenMP and MPI [26].

4.1.1 Hybrid Tetrahedral Grids

Since *HyTeG* is centered around multigrid, its algorithms work with a hierarchy of nested grids. This grid hierarchy is obtained by a succession of uniform refinements. The basis for refinement is a coarse tetrahedral mesh, which we say is on *level 0*. Each tetrahedron in the mesh is individually divided into eight smaller tetrahedra of equal volume which together form level 1. This procedure is repeated until a sufficiently fine mesh is obtained.

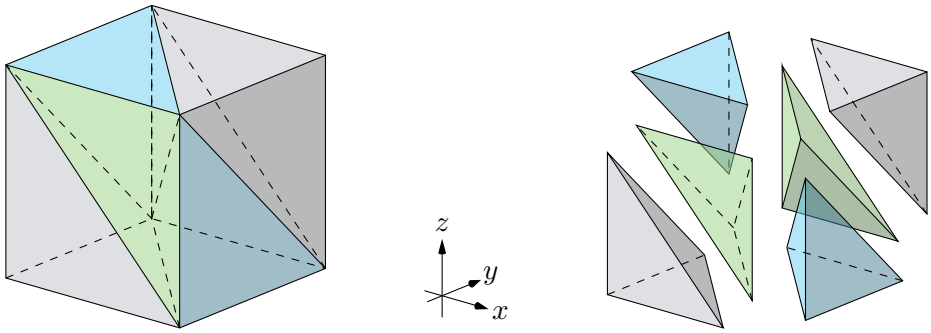
Figure 4.1 illustrates the refinement of the reference tetrahedron. First, all edges are subdivided at their respective centers which can alternatively be thought of as cutting off the corners of the tetrahedron. The result are four smaller tetrahedra which are scaled down versions of the original solid and a central octahedron. This octahedron is further divided into four tetrahedra by connecting two opposite corners with an additional edge. Here, three possible choices can be made. In *HyTeG*, the edge

[1] Available at `https://i10git.cs.fau.de/hyteg/hyteg`.



(a) Coarse tetrahedron. (b) Subdivision of all edges. (c) Subdivision of internal octahedron.

Figure 4.1: Octasection of a single tetrahedron into equivolume tetrahedra.



(a) Packed. (b) Exploded.

Figure 4.2: All six cell types arise from the division of a cube. Tetrahedra of the same color are similar.

drawn in Figure 4.1c is chosen. After refinement, the tetrahedron has been divided into eight smaller tetrahedra of equal volume which can be sorted into three different similarity classes. Remarkably, successive refinement does not increase the number of similarity classes. Moreover, on each level $l \geq 2$, each similarity class is made up of two congruence classes. In *HyTeG*, the six different congruence classes are referred to as *cell types*.

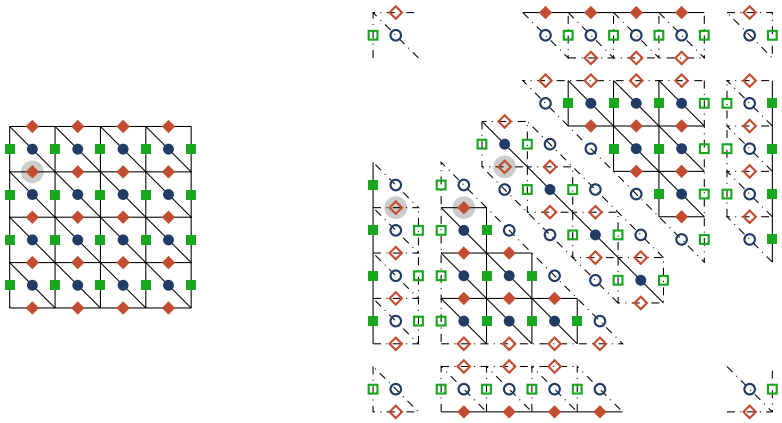
The inside of a refined tetrahedron can also be viewed as being built from equal cubes, where each cube is formed by taking one cell of each type. This is depicted in Figure 4.2. More information about this refinement strategy can be found in [29, 30].

Similar to cells, edges are classified into seven different *edge types* corresponding to the direction into which an edge is pointing. The edge types/directions are labelled X, Y, Z, XY, XZ, YZ and XYZ. For affine cells this classification is based on how the edges are directed when transformed to the reference tetrahedron.

The result of the mesh refinement is a block-structured mesh. It combines the flexibility of an unstructured mesh with the performance benefits of a structured mesh. For example, the uniform nature of each block can be exploited to achieve better node-level performance. In particular, it allows index-based and indirection-free memory accesses and lets matrix operations be implemented through stencil based code. This leads to less memory and memory bandwidth consumption and lays the foundation for extreme simulation sizes [26].

4.1.2 Storage of Degrees of Freedom

Data in *HyTeG* is stored on *macro-primitives*: cells, faces, edges and vertices of the coarse mesh. The primitives form a fully distributed container data structure for arbitrary data [26]. Of course, this includes degrees of freedom (DoFs). In addition to the DoFs which are physically located on the respective part of the mesh, each macro-primitive stores copies of directly neighboring DoFs in *ghost layers*. Every time an operation needs values from neighboring primitives, these ghost layers must be updated through communication [26]. As an example, Figure 4.3 pictures the distribution of edge DoF data on a mesh in two dimensions.



(a) A square mesh on level 2 with DoFs on the edges.

(b) Partitioning of the domain into macro-faces, macro-edges and macro-vertices.

Figure 4.3: A 2D illustration of *HyTeG*'s data distribution concept. DoFs are stored on the edges in this example. Horizontal, vertical and diagonal DoFs are distinguished and grouped together. On the right-hand side, the data resident on each macro-primitive is shown. The highlighted unknown is owned by the face and is additionally resident in the ghost layers of the neighboring edges.

HyTeG provides reusable building blocks which allow placing DoFs on nodes, edges, faces or volumes of the mesh. These can be combined to implement various finite

element discretizations of desired degree. At the time of this writing, continuous piecewise Lagrangian polynomials of degrees 0, 1 and 2 and discontinuous functions with piecewise constant or linear Lagrange basis functions are implemented. In the following, we describe how we add support for the space $\mathcal{P}_1^- \Lambda^1$ of Nédélec edge elements of type I and order 1.

4.2 The Nédélec Space $\mathcal{P}_1^- \Lambda^1$

The implementation of the Nédélec space is centered around `N1E1VectorFunction`, a C++ class representing a function in $\mathcal{P}_1^- \Lambda^1$. The nomenclature follows the notation used in [31]. An `N1E1VectorFunction` internally uses an `EdgeDoFFunction` to store its DoFs and provides methods for projecting a continuous function into $\mathcal{P}_1^- \Lambda^1$, evaluating the function at given coordinates and communicating interface values between macro-primitives.

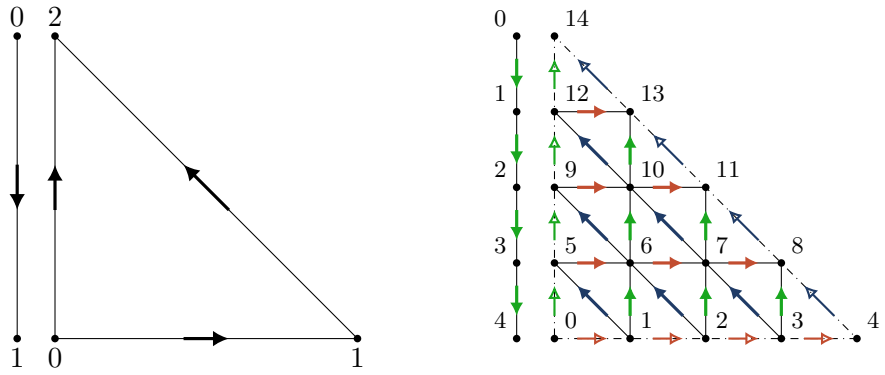
4.2.1 Edge Orientation Considerations

Recall that the DoFs of a function in $\mathcal{P}_1^- \Lambda^1(\mathcal{T}_h)$ depend on the tangent vectors of the edges in the mesh $\mathcal{T}_h$. The direction of an edge that was introduced before tells us e.g. that an edge (after transforming to the reference tetrahedron) is parallel to the x-axis. However, this determines the tangent vector only up to the sign. In addition to the direction, we must know the *orientation* of an edge, i.e. whether it points to the left or right. Since a single DoF is adjacent to several elements, care has to be taken that edge orientations are taken into account consistently. To ensure this, we follow different strategies inside and across macro-primitives.

Inside Macro-Primitives

Inside macro-primitives the orientation of each edge is uniquely determined by the local order of the adjacent macro-vertices. This obviously results in consistent orientations as the macro-vertices are independent of specific micro-elements. Making the orientation dependent on the *local*, in contrast to the *global*, vertex indexing has the benefit that edge orientations are the same on each macro-tetrahedron. This eases the implementation of some routines presented below.

The local indices of macro-vertices range from 0 to 3 on macro-cells and are prescribed by the mesh file. Based on these indices, we lexicographically assign a local index to each micro-vertex (cf. Figure 4.4). This vertex ordering then induces the edge orientation: an edge points from the vertex with the lower index towards the vertex with the higher index. Owing to the lexicographic ordering, all except XYZ-micro-edges share their orientation with the orientation of the parallel macro-edge. Given



(a) Neighboring macro-primitives and their local macro-vertex indexing.

(b) Micro-vertex numbering and micro-edge orientations induced by macro-orientation.

Figure 4.4: A macro-face and one neighboring macro-edge. In this example their edge orientations do not align. Macro-vertex indexing induces a lexicographic micro-vertex indexing and edge orientation from lower to higher index. Ghost layers of the edge are not shown for clarity.

that there is no XYZ-edge on the macro-level, only the characterization through micro-vertices can be applied.

Across Macro-Primitives

Since the macro-vertex ordering determines the direction of the XYZ-edge during refinement, it also influences the quality of the mesh. Of course, it is desirable not to eliminate this tuning parameter and therefore edges may point in opposite directions on neighboring primitives. This is exemplified in Figure 4.4.

As a result, we must correct for the mismatching orientation on neighboring primitives. Note that even though the orientations of edges might differ, their locations and directions are still the same. According to (2.48), the sign of a DoF changes when the orientation of an edge is flipped. Similarly, by going back to the derivation of the affine basis functions (2.51), it is easy to verify that flipping an edge also flips the sign of the associated basis function while keeping the remaining functions unaltered. Therefore, edge orientation can be accounted for by changing the sign of the respective DoF.

The corresponding logic is implemented in the C++ classes `N1E1PackInfo` and `N1E1AdditivePackInfo`. These are responsible for (de-)serializing data that is to be exchanged between macro-primitives. If the two primitives reside on different processors, the data is serialized into a buffer and sent via MPI. Otherwise, the data is copied between the data arrays of the primitives directly. In either case, the correct signs of the DoFs are determined on the higher dimensional primitive.

`N1E1PackInfo` is used whenever the ghost layers of primitives must be updated. This is necessary to perform most stencil operations that operate directly on DoFs. In the process of copying unknowns which are owned by a particular macro-primitive to the ghost layers of the neighboring primitives, previously stored values are overwritten. On the other hand, `N1E1AdditivePackInfo` reads values from the ghost layers of the neighboring macro-primitives and adds them to the owned values. This type of communication is a useful building block for routines that operate on the individual elements of the triangulation.

Lastly, it is worth pointing out that some vectors require "normal" communication, i.e. the signs of their entries should not be adjusted. This is the case for data that is not actually a member of $\mathcal{P}_1^- \Lambda^1$ mathematically speaking but is required to be of the same C++ class `N1E1VectorFunction` due to *HyTeG*'s design. Examples for such data are matrix diagonals and vectors that assign to each unknown a unique index. The latter are needed for interfacing with PETSc. The implementation supports these cases by giving access to the underlying `EdgeDoFFunction`. By executing communication routines directly on the `EdgeDoFFunction`, data is transferred to the neighboring primitives as is.

4.2.2 Projection

When projecting a continuous function $\boldsymbol{u}$ into $\mathcal{P}_1^- \Lambda^1(\mathcal{T}_h)$, every DoF is determined according to

$$\boldsymbol{u} \mapsto \boldsymbol{u}(\boldsymbol{m}) \cdot |e| \boldsymbol{t}_e, \quad e \in \triangle_1(\mathcal{T}_h)\,, \tag{4.1}$$

where $\boldsymbol{m}$, $|e|$ and $\boldsymbol{t}_e$ are the midpoint, length and unit tangent of the edge e, respectively. Note that this is the midpoint rule approximation of the integral (2.48). The midpoint quadrature is chosen since it is exact for the entirety of $\mathcal{P}_1^- \Lambda^1$ and has the same consistency order as the discretization.

To verify that there is no reason to resort to more accurate quadrature rules, we assess the accuracy of the projection operator. To that end, we mesh the unit cube $\Omega = (0,1)^3$ with 24 macro-tetrahedra and project a sinusoidal function ((3.22) with $k = 4$) with different quadrature rules. Specifically, we try Gauss-Legendre quadrature with $n \in \{1, 2, 3\}$ points which is exact for polynomials of degree $2n - 1$ [32, p. 192]. For $n = 1$ this is the midpoint rule. After projecting the function to the finite element space, we evaluate it at 1000 random points in the domain and determine the average Euclidean norm of the error. In all three cases, we get essentially the same result (cf. Table 4.1).

Since the term $|e|\boldsymbol{t}_e$ is constant for each edge type, it is only calculated once on each macro-primitive. Similarly, there is a specialization for constant $\boldsymbol{u}$ which evaluates (4.1) only once for each edge type.

Level \ n	1	2	3
3	$3.216 \cdot 10^{-1}$	$3.248 \cdot 10^{-1}$	$3.247 \cdot 10^{-1}$
4	$1.678 \cdot 10^{-1}$	$1.686 \cdot 10^{-1}$	$1.686 \cdot 10^{-1}$
5	$8.250 \cdot 10^{-2}$	$8.241 \cdot 10^{-2}$	$8.241 \cdot 10^{-2}$
6	$4.080 \cdot 10^{-2}$	$4.078 \cdot 10^{-2}$	$4.078 \cdot 10^{-2}$
7	$2.079 \cdot 10^{-2}$	$2.079 \cdot 10^{-2}$	$2.079 \cdot 10^{-2}$

Table 4.1: Average Euclidean norm of the error of the projection operator approximated by Gauss-Legendre quadrature with n points.

4.2.3 Evaluation

Evaluating an `N1E1VectorFunction` is implemented as described at the end of Chapter 2. First, the function is evaluated with respect to the basis on the reference tetrahedron. Then, the result is Piola transformed to the respective affine micro-element. The computationally most costly part of this operation is determining the micro-element which contains the given coordinates. However, the micro-element might already be known by the calling code. One example is the VTK output routine which evaluates the function at the center of every micro-element. Therefore, the micro-element can optionally be passed as an argument.

4.2.4 Operator Application

Let V be a Hilbert space, $V_h \subset V$ a finite element subspace with basis $\{\phi_i\}$ (in our case $V = \boldsymbol{H}(\mathbf{curl})$, $V_h = \mathcal{P}_1^- \Lambda^1(\mathcal{T}_h)$) and $a(\cdot,\cdot) : V_h \times V_h \to \mathbb{R}$ a bilinear form defined by $a(u,v) := \int_\Omega \psi(u,v)\,\mathrm{d}\boldsymbol{x}$, where ψ is some integrand. Then, $a(\cdot,\cdot)$ has a linear operator $A \in \mathbb{R}^n \times \mathbb{R}^n$ associated to it, where

$$a_{ij} = a(\phi_j, \phi_i) = \int_\Omega \psi(\phi_j, \phi_i)\,\mathrm{d}\boldsymbol{x}, \quad i,j = 1, \dots n\,. \tag{4.2}$$

When solving a variational problem of the form

$$\text{find } u \in V, \text{ such that } a(u,v) = f(v), \quad \text{for all } v \in V\,, \tag{4.3}$$

A is the system matrix of the resulting discretized linear system of equations (LSE) [33, Theorem 8.5].

Our implementation makes it possible to apply any operator arising from a bilinear form to a Nédélec function. It makes use of the fact that the integrals in (4.2) can be written as a sum over the elements $T \in \mathcal{T}_h$ [21, p. 92][33, p. 216]:

$$a_{ij} = \int_\Omega \psi(\phi_j, \phi_i)\,\mathrm{d}\boldsymbol{x} \;= \sum_{T \in \mathcal{T}_h} \int_T \psi(\phi_j, \phi_i)\,\mathrm{d}\boldsymbol{x}\,. \tag{4.4}$$

The integration over a single element yields an *element matrix* which has the same number of rows and columns as there are DoFs on the element (6×6 in the case of $\mathcal{P}_1^- \Lambda^1$ on tetrahedral meshes). The assembly of the element matrices is separated from the remaining code so that most of the implementation can be reused for all bilinear forms.

Even though the mesh is uniformly refined into micro-elements of just six congruence classes, the element matrices are not assumed to be constant for each element type. This way, bilinear forms with spatially varying coefficients are supported. Furthermore, this is a prerequisite for the implementation of non-affine transformations between a physical and computational coordinate system.

Per default not more than one element matrix is held in memory at a time. Instead, linearity of the matrix vector product $A\boldsymbol{x} = \boldsymbol{b}$ is exploited and $\boldsymbol{x}$ is multiplied with the element matrices directly. This leads to very low memory requirements and enables extreme scalability. However, it is possible to precompute and store all element matrices for faster repeated operator applications. This is favorable in cases in which memory (bandwidth) is not the bottleneck, e.g. because the system is not very large. Moreover, the inverse diagonal entries are precomputed and stored as a vector. This allows for efficient application of the Chebyshev smoother.

Bilinear Forms

All bilinear form specific behavior is implemented in separate C++ classes. Their sole responsibility is to determine the element matrices by evaluating the integral in (4.4). We provide implementations for the mass form $a_m(\boldsymbol{\phi}_j, \boldsymbol{\phi}_i) = \int_T \boldsymbol{\phi}_j \cdot \boldsymbol{\phi}_i \,\mathrm{d}\boldsymbol{x}$ and the **curl-curl** form $a_c(\boldsymbol{\phi}_j, \boldsymbol{\phi}_i) = \int_T \mathbf{curl}\, \boldsymbol{\phi}_j \cdot \mathbf{curl}\, \boldsymbol{\phi}_i \,\mathrm{d}\boldsymbol{x}$. Again, we make use of the affine invariance of $\mathcal{P}_1^- \Lambda^1$ discussed at the end of Chapter 2 and calculate the integrals over the reference tetrahedron $\hat{T}$. In the following, F denotes an affine map which maps the reference tetrahedron to the affine tetrahedron T:

$$F : \hat{T} \to T, \quad \hat{\boldsymbol{x}} \mapsto B\hat{\boldsymbol{x}} + \boldsymbol{b}\,, \tag{4.5}$$

where we write $\hat{\boldsymbol{x}}$ for a vector in the reference coordinate system. Note that the Jacobian of F is simply B.

Mass Form Using the Piola transform (2.54) we can express the basis functions on the affine element as $\boldsymbol{\phi}_i = B^{-T}(\hat{\boldsymbol{\phi}}_i \circ F^{-1})$ with respect to the reference basis function $\hat{\boldsymbol{\phi}}_i$. Inserting this into the mass-form yields

$$\int_T \boldsymbol{\phi}_j \cdot \boldsymbol{\phi}_i \,\mathrm{d}\boldsymbol{x} = |\det B| \int_{\hat{T}} \left(B^{-T}\hat{\boldsymbol{\phi}}_j\right)^T \left(B^{-T}\hat{\boldsymbol{\phi}}_i\right) \mathrm{d}\hat{\boldsymbol{x}}\,. \tag{4.6}$$

Evaluating (4.6) requires calculation of the inverse of B. There are different ways to handle the inverse and since it is not clear how this affects the numerical stability, we benchmark three different approaches:

1. Compute B^{-1} numerically using cofactors. Insert the result into a formula for (4.6) obtained by symbolic integration using SymPy [34].
2. Solve (4.6) symbolically as is (depending on B not on B^{-1} as in approach 1) and apply symbolic simplifications and common subexpression elimination (CSE) to keep the resulting formula manageable.
3. Determine an LU-decomposition of B with full pivoting numerically. Reformulate (4.6) to pull B^{-1} out of the integral so that all multiplications with the inverse can be replaced by forward-backward-solves with the LU-decomposition. For this we rewrite $\hat{\boldsymbol{\phi}}_i =: A_i(x, y, z, 1)^T =: A_i \boldsymbol{z}$ and abbreviate $C_i := B^{-T} A_i \in \mathbb{R}^{3\times 4}$. Now we seek to separate C_i from the integral:

$$\int_{\hat{T}} \left(B^{-T}\hat{\boldsymbol{\phi}}_j\right)^T \left(B^{-T}\hat{\boldsymbol{\phi}}_i\right) \mathrm{d}\hat{\boldsymbol{x}} = \int_{\hat{T}} \left(B^{-T} A_j \boldsymbol{z}\right)^T \left(B^{-T} A_i \boldsymbol{z}\right) \mathrm{d}\hat{\boldsymbol{x}} \tag{4.7a}$$

$$= \int_{\hat{T}} \boldsymbol{z}^T C_j^T C_i \boldsymbol{z} \, \mathrm{d}\hat{\boldsymbol{x}} \tag{4.7b}$$

$$= \int_{\hat{T}} \sum_{l=1}^{3} \sum_{k=1}^{4} \sum_{m=1}^{4} z_k c_{lk}^j c_{lm}^i z_m \, \mathrm{d}\hat{\boldsymbol{x}} \tag{4.7c}$$

$$= \int_{\hat{T}} \sum_{l=1}^{3} \sum_{k=1}^{4} c_{lk}^j \left(C_i \boldsymbol{z}\boldsymbol{z}^T\right)_{lk} \mathrm{d}\hat{\boldsymbol{x}} \tag{4.7d}$$

$$= \sum_{l=1}^{3} \sum_{k=1}^{4} c_{lk}^j \left(C_i \int_{\hat{T}} \boldsymbol{z}\boldsymbol{z}^T \, \mathrm{d}\hat{\boldsymbol{x}}\right)_{lk}, \tag{4.7e}$$

where the integral in (4.7e) is to be understood componentwise and evaluates to

$$\int_{\hat{T}} \boldsymbol{z}\boldsymbol{z}^T \, \mathrm{d}\hat{\boldsymbol{x}} = \frac{1}{120} \begin{pmatrix} 2 & 1 & 1 & 5 \\ 1 & 2 & 1 & 5 \\ 1 & 1 & 2 & 5 \\ 5 & 5 & 5 & 20 \end{pmatrix}. \tag{4.8}$$

To compare the numerical stability of the three approaches, we determine the discrepancy in the element-matrix when using single and double precision. To that end, we pick 1000 elements which have one vertex fixed at the origin and the others randomly distributed on the unit sphere. On each element, we compute the element matrices in single and double precision, and calculate the Frobenius norm of the difference. All three approaches perform very similarly. The maximum difference in the Frobenius norm is on the order 1 and the average over the remaining 999 tetrahedra is on the order 10^{-6}. Since this experiment is inconclusive, we select approach 1 based on performance measurements.

curl-curl Form In order to transform a_c we use (4.9) to map the **curl** to the reference coordinate system [35]:

$$\mathbf{curl}\, \boldsymbol{\phi}_i = \frac{1}{\det B} B \left(\widehat{\mathbf{curl}}\, \hat{\boldsymbol{\phi}}_i \circ F^{-1}\right). \tag{4.9}$$

The **curl-curl**-form becomes:

$$\int_T \mathbf{curl}\,\boldsymbol{\phi}_j \cdot \mathbf{curl}\,\boldsymbol{\phi}_i \,\mathrm{d}\boldsymbol{x} = \int_{\hat{T}} \frac{1}{|\det B|} \left(B\widehat{\mathbf{curl}}\,\hat{\boldsymbol{\phi}}_j\right)^T \left(B\widehat{\mathbf{curl}}\,\hat{\boldsymbol{\phi}}_i\right) \mathrm{d}\hat{\boldsymbol{x}}\,. \tag{4.10}$$

Note that in the case of linear finite elements the functions $\widehat{\mathbf{curl}}\,\hat{\boldsymbol{\phi}}_i$ are constant, which makes the evaluation of (4.10) particularly easy.

Linear Form

The assembly of an LSE also includes the determination of the right-hand side (RHS) from a linear form

$$b(\boldsymbol{\phi}_i) = \sum_{T\in\mathcal{T}_h} \int_T \boldsymbol{f}\cdot\boldsymbol{\phi}_i \,\mathrm{d}\boldsymbol{x} = \sum_{T\in\mathcal{T}_h} |\det B| \int_{\hat{T}} (\boldsymbol{f}\circ F)\cdot(B^{-T}\hat{\boldsymbol{\phi}}_i)\,\mathrm{d}\hat{\boldsymbol{x}}\,. \tag{4.11}$$

Since $\boldsymbol{f}$ is spatially dependent and only known at runtime, we can not rely on symbolic integration of $\boldsymbol{f}$. Instead, we use the quadrature rule of degree 6 by Xiao and Gimbutas [36] for the integral (4.11). Again, code generation is employed to ease and optimize the implementation of the quadrature.

Because it is convenient to reuse the code developed for bilinear forms, linear forms are implemented as bilinear forms where only the diagonal of the element matrices is non-zero. The RHS-vector is then obtained as the diagonal of the resulting matrix.

4.3 Multigrid

We now turn our attention to the implementation of the multigrid scheme, in particular the grid transfer operators and the hybrid smoother.

4.3.1 Grid Transfer Operators

Prolongation

The Prolongation operator is compactly defined by the natural embedding of the coarse grid function space into the fine grid space: $\mathcal{P}_1^-\Lambda^1(\mathcal{T}_{2h}) \subset \mathcal{P}_1^-\Lambda^1(\mathcal{T}_h)$. Practically speaking, this means that we can find the DoFs on the fine level by projecting the coarse grid function to the fine level. From the second characterization it is easier to deduce a local relation which can be used to obtain the fine grid DoFs from the surrounding unknowns on the coarse grid. On each element $T_{2h} \in \mathcal{T}_{2h}$, a coarse grid function $\boldsymbol{u}_{2h}$ has the representation $\boldsymbol{u}_{2h}(T_{2h}) = \sum_{j=0}^{5} u_j^{(2h)} \boldsymbol{\phi}_j^{(2h)}$, where the index j enumerates the edges of T_{2h} and $u_j^{(2h)}$ are the DoFs on these edges. By selecting a

coarse element T_{2h} which contains the i-th DoF $u_i^{(h)}$ on the fine level and projecting $\boldsymbol{u}_{2h}(T_{2h})$ according to (2.48) we get an equation for the prolongation:

$$u_i^{(h)} = \int_{e_i^{(h)}} \left(\sum_{j=0}^{5} u_j^{(2h)} \boldsymbol{\phi}_j^{(2h)} \right) \cdot \boldsymbol{t}_{e_i^{(h)}} \,\mathrm{d}\boldsymbol{x} = \sum_{j=0}^{5} \int_{e_i^{(h)}} \boldsymbol{\phi}_j^{(2h)} \cdot \boldsymbol{t}_{e_i^{(h)}} \,\mathrm{d}\boldsymbol{x}\, u_j^{(2h)} \,. \tag{4.12}$$

To further simplify (4.12), we need the following two results.

Proposition 4.1. *Let $F : \hat{T} \to T, \hat{\boldsymbol{x}} \mapsto B\hat{\boldsymbol{x}} + b$ be an affine mapping from a reference tetrahedron $\hat{T}$ to an affine element T. Further, let $\boldsymbol{\phi}$ be a vector function on T that arises from the pullback (2.54) of $\hat{\boldsymbol{\phi}}$, a function over $\hat{T}$. Projecting $\hat{\boldsymbol{\phi}}$ to $\mathcal{P}_1^- \Lambda^1(\hat{T})$ and $\boldsymbol{\phi}$ to $\mathcal{P}_1^- \Lambda^1(T)$ yields the same DoFs.*

Proof. The proposition follows from the transformation of the integral:

$$\int_{e_i} \boldsymbol{\phi} \cdot \boldsymbol{t}_{e_i} \,\mathrm{d}\boldsymbol{x} \tag{4.13a}$$

$$= \int_{\hat{e}_i} \left(B^{-T} \hat{\boldsymbol{\phi}} \right)^T \left(\frac{1}{\det B} B \boldsymbol{t}_{\hat{e}_i} \right) \mathrm{d}\hat{\boldsymbol{x}} \det B \tag{4.13b}$$

$$= \int_{\hat{e}_i} \hat{\boldsymbol{\phi}} \cdot \boldsymbol{t}_{\hat{e}_i} \,\mathrm{d}\hat{\boldsymbol{x}} \,, \tag{4.13c}$$

where in (4.13b) we must divide by $\det B$ so that $\boldsymbol{t}_{\hat{e}_i^{(h)}}$ keeps its unit length. From comparing (4.13a) and (4.13c), we learn that we can choose the coordinate system freely. □

Proposition 4.2. *Let $\boldsymbol{\phi}_{ij}$ be the basis function of $\mathcal{P}_1^- \Lambda^1(T)$ attached to the edge e_{ij} which is directed from vertex i to j. We refer to the face opposite to vertex i as f_i. The restriction of $\boldsymbol{\phi}_{ij}$ to one of the opposing faces (f_i, f_j) is orthogonal to that face: $\boldsymbol{\phi}_{ij}(f_i) \perp f_i$ and $\boldsymbol{\phi}_{ij}(f_j) \perp f_j$.*

Proof. Going back to (2.55) the basis function $\boldsymbol{\phi}_{ij}$ can be written in barycentric coordinates as $\boldsymbol{\phi}_{ij} = \lambda_i \,\mathbf{grad}\, \lambda_j - \lambda_j \,\mathbf{grad}\, \lambda_i$. Furthermore, $\lambda_i = 0$ on f_i and $\mathbf{grad}\, \lambda_i$ is the inward facing normal vector of f_i. The proposition for f_i follows directly. The case f_j is completely analogous. □

Proposition 4.2 tells us that in (4.12) those terms vanish for which the fine grid edge is contained in one of the opposing faces of the coarse grid edge. Phrased differently, if the fine grid edge lies on a coarse grid edge, only the term involving this coarse grid edge is non-zero. If on the other hand, the fine grid edge is located on the inside of a coarse grid face, it suffices to sum over the three surrounding coarse grid edges. Only if the fine grid edge is located inside a coarse grid tetrahedron, all six surrounding coarse grid edges must be considered. Together with tangential continuity of the

Nédélec-space it follows that the choice of T_{2h} in (4.12) does not matter. Moreover, the prolongation operator $I_{2h}^h : \mathcal{P}_1^- \Lambda^1(\mathcal{T}_{2h}) \to \mathcal{P}_1^- \Lambda^1(\mathcal{T}_h)$ can be written as

$$u_h = I_{2h}^h u_{2h}, \quad \left(I_{2h}^h\right)_{i,j} := \int_{e_i^{(h)}} \boldsymbol{\phi}_j^{(2h)} \cdot \boldsymbol{t}_{e_i^{(h)}} \, \mathrm{d}\boldsymbol{x} \,, \tag{4.14}$$

where $\boldsymbol{\phi}_j^{(2h)}$ is one of the basis functions attached to the coarse grid edge $e_j^{(2h)}$.

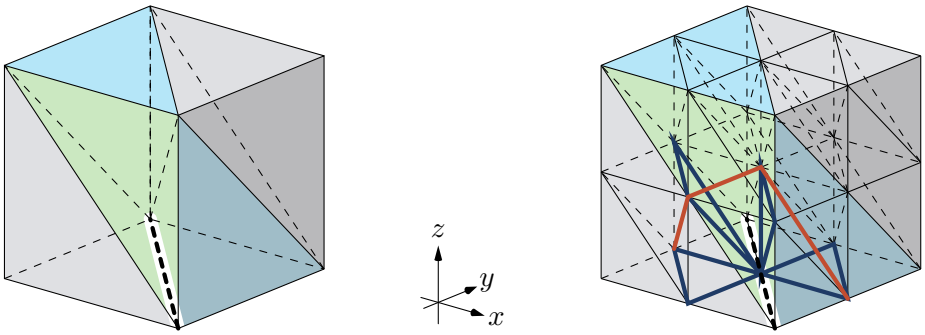
In light of Proposition 4.1, the prolongation operator is invariant under affine transformations. Therefore, we restrict the following discussion to refinements of the reference tetrahedron. What remains is finding stencil entries that implement (4.14) in a matrix-free fashion.

By *stencil entry* we mean a tuple (w, o, d_{2h}, d_h) consisting of a weight w, an index offset o and coarse/fine edge directions d. The application of the prolongation operator (4.14) can be decomposed into sums of the form

$$u_{(2(x,y,z)+o,d_h)}^{(h)} = \sum w u_{((x,y,z),d_{2h})}^{(2h)} \,, \tag{4.15}$$

where $u_{((x,y,z),d)}$ refers to the DoF associated to the edge with the three-dimensional index $(x, y, z) \in \mathbb{N}_0^3$ and direction d, and the sum runs over all coarse grid edges.

While owing to Proposition 4.1 the weights are the same for all cell types, the index offsets are not. This is due to the different orientations of the cell types inside a macro-element. Therefore, it is easiest to tile the mesh by cubes as pictured in Figure 4.5. It is enough to look at this cube, since all cell types and possible combinations of coarse and fine edges are contained in it. What is more, it is easy to iterate over instances of this cube. Note, however, that in proximity of the slanted face of the macro-tetrahedron, some elements inside the cube are missing.



(a) Coarse Level. (b) Fine Level.

Figure 4.5: A cube and its coarse and fine triangulation. As an example the diagonal coarse grid edge on the bottom side is considered. On the fine level all edges possibly leading to non-zero stencil entries are highlighted (dashed black: same edge, blue: adjacent faces, red: adjacent cells).

Below, we determine all stencil entries which make up I_{2h}^h by considering all coarse/fine edge pairs in a single cube. The general procedure is given in Algorithm 4.1. First, in Lines 3 to 5, for each of the 19 coarse grid edges, we collect the edges on the fine grid for which $(I_{2h}^h)_{i,j}$ might be non-zero. As discussed above, these are the inner edge of each adjacent cell (marked red in Figure 4.5b), the three inner edges of each adjacent face (marked blue in Figure 4.5b) and the two fine edges at the same location as the coarse edge (marked black in Figure 4.5b). So far, there is not a one-to-one mapping between coarse/fine edge pairs and non-zero stencil entries. On one hand, some of the above edge combinations yield zero-valued stencil entries. On the other hand, certain coarse/fine edge pairs give the same stencil entry. For instance, every pair that lies entirely on one of the sides of the cube, appears equally on the opposing side.

Next, in Lines 7 to 9, the stencil entry corresponding to the current edge pair is determined. First, the weight is calculated through (4.14). Since we only consider the linear space $\mathcal{P}_1^- \Lambda^1$, the integral can be conveniently evaluated using the midpoint rule. Subsequently, the offset between the indices of the coarse and fine edge is determined. Finally, duplicate stencil entries are filtered out by comparing index offset and edge directions. Fresh stencil entries are appended to the list of all entries (Line 11).

Algorithm 4.1: Calculation of prolongation stencil.

Output: stencilEntries

```
1  stencilEntries ← empty list
2  foreach coarseEdge in cube do
3  |  cells ← CellsAdjacentTo(coarseEdge)
4  |  faces ← FacesAdjacentTo(coarseEdge)
5  |  fineEdges ← InnerFineEdgesOf(cells) and InnerFineEdgesOf(faces) and
   |   FineEdgesOf(coarseEdge)
6  |  foreach fineEdge in fineEdges do
7  |  |  weight ← Evaluate(4.14)(coarseEdge, fineEdge)
8  |  |  indexOffset ← IndexOffset(coarseEdge, fineEdge)
9  |  |  stencilEntry ← (weight, indexOffset, DirectionOf(coarseEdge),
   |  |   DirectionOf(fineEdge))
10 |  |  if not stencilEntry in stencilEntries then
11 |  |  |  insert stencilEntry into stencilEntries
12 |  |  end
13 |  end
14 end
15 return stencilEntries
```

Algorithm 4.1 is not executed at runtime but rather used to generate the stencil code for the application of the prolongation operator. This leads to a very efficient implementation since, apart from the inevitable summation of unknowns, only loop logic and index bound checks must be performed at runtime. Furthermore, it allows to

sort the stencil entries in such a way that accesses to the DoFs are optimized according to the layout of unknowns in memory.

From Proposition 4.2 it follows that DoFs on the inside of cells do not influence unknowns on the boundary. Therefore, communication towards lower dimensional primitives can be forgone. In addition, we avoid communication during prolongation by implementing kernels on all primitive types (macro-edges, -faces and -cells but not macro-vertices since they do not store unknowns). On macro-edges prolongating is completely local and therefore communication-free. On macro-faces and -cells ghost layers from neighboring primitives of lower dimension must be updated before prolongating. In case of macro-faces these are edges and for macro-cells edges and faces. After the prolongation operation, no additional communication is necessary. Therefore, the communication volume is less than half of that needed for applying an operator arising from a bilinear form.

Restriction

The restriction operator is defined as the transpose of the prolongation as is usual in the context of Galerkin coarsening [3, p. 274]: $I_h^{2h} := (I_{2h}^h)^T$. Therefore, the stencil entries from the prolongation operator can be reused for the restriction. The fundamental difference between the two operators is that the restriction operator reads values from fine grid edges and sums them according to the stencils entries to obtain the unknowns on the coarse grid. Due to the reversed stencil directions, both operators have distinct communication patterns. In case of the restriction, values on the inside of cells are independent of boundary DoFs. Consequently, communication towards higher dimensional primitives is not necessary.

The first step in applying the restriction operator is setting the ghost layers on the coarse level to zero on all primitives. Next, the stencil operation is performed. In this process, only inner/owned DoFs are read since values in the ghost layers are not up-to-date. However, all unknowns including those on the boundary are written to. So far, all operations are purely local and in fact, the owned DoFs of the cells are already correctly determined. On the lower dimensional primitives, however, contributions from the respectively higher dimensional primitives must be added to the local contributions. Therefore, three additive communication steps are carried out: from faces to edges, from cells to faces and from cells to edges.

Gradient

It is a known fact that the gradient operator from $\mathcal{P}_1^- \Lambda^0$ to $\mathcal{P}_1^- \Lambda^1$ can be easily applied [8]. Nevertheless, we give a derivation of this result in the following.

Let $u \in \mathcal{P}_1^- \Lambda^0(T)$ be a scalar valued, piecewise linear function. It enjoys a representation with respect to the Lagrangian basis $\psi_i, i = 1 \dots 4$ with four DoFs u_i. The

gradient of u on a simplex $T = B\hat{T} + \boldsymbol{b}$ is then given as transformation from the reference tetrahedron:

$$\mathbf{grad}\, u = \mathbf{grad} \sum_{i=1}^{4} u_i \psi_i = \sum_{i=1}^{4} u_i \,\mathbf{grad}\, \psi_i \tag{4.16a}$$

$$= \frac{1}{\det B} B^{-T} \begin{pmatrix} -1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{pmatrix} = \frac{1}{\det B} B^{-T} \begin{pmatrix} u_2 - u_1 \\ u_3 - u_1 \\ u_4 - u_1 \end{pmatrix} . \tag{4.16b}$$

When viewed as an element of $\mathcal{P}_1^- \Lambda^1$ the DoFs of $\mathbf{grad}\, u$ are:

$$\int_e \mathbf{grad}\, u(\boldsymbol{x}) \cdot \boldsymbol{t} \,\mathrm{d}\boldsymbol{x} = \int_e \frac{1}{\det B} B^{-T} \begin{pmatrix} u_2 - u_1 \\ u_3 - u_1 \\ u_4 - u_1 \end{pmatrix} \cdot \boldsymbol{t} \,\mathrm{d}\boldsymbol{x} = \int_{\hat{e}} \begin{pmatrix} u_2 - u_1 \\ u_3 - u_1 \\ u_4 - u_1 \end{pmatrix} \cdot \hat{\boldsymbol{t}} \,\mathrm{d}\hat{\boldsymbol{x}} . \tag{4.17}$$

Transforming the integral to the reference tetrahedron $\hat{T}$ in (4.17) reveals a particularly convenient implementation of the gradient operator. The DoF of the gradient associated to edge e_{ij} from vertex i to j is simply $u_j - u_i$. Since the procedure is the same for every edge, i.e. does not depend on the direction of the edge or the types of the neighboring cells, the implementation is much simpler than that of the prolongation or restriction operator.

Regardless, communication must be performed in essentially the same way as for the prolongation. This is because edge DoFs on the boundary of cells are not connected to vertex DoFs located inside of cells. Concretely, this means that each primitive needs to update the ghost layers only from neighboring primitives of a lower dimension. Unlike the prolongation code, the gradient operator must also communicate data from the macro-vertices to the other primitives. Once the ghost layers are up-to-date, the gradient can be determined locally and no further communication is necessary.

Lifting

The lifting operator is the transpose of the gradient. Thus, it can be implemented by summing at each vertex the unknowns of the connected edges. In this procedure, DoFs of outgoing edges are weighted negatively. Analogous to the implementation of the restriction operator, on each macro-primitive no edge DoF data is read from the ghost layers but vertex-DoFs are written to ghost layers. After the computations, the partial sums of the interface primitives are added together by performing additive communication from macro-primitives to their respective neighbors of lower dimension.

4.3.2 Hybrid Smoother

The implementation of the hybrid smoother is a direct translation of Algorithm 3.2. The smoothers in the Nédélec and potential space can be configured on the caller side.

Throughout this work, we use the Gauss-Seidel method for smoothing in potential space. In *HyTeG*'s implementation, the unknowns are updated in a block-wise fashion. First, the DoFs resident on macro-vertices are relaxed. The result is communicated to macro-edges where relaxation is carried on. After propagating the results to macro-faces, smoothing is continued there and eventually on macro-cells. Since we only deal with constant coefficients, the stencil weights of the operator are constant on each macro-primitive. Therefore, they are computed only once and stored without requiring large amounts of memory.

Smoothing in the Nédélec space is performed with the Chebyshev smoother. We use power iteration to approximate the spectral radius of the scaled system matrix. From this we obtain estimates for the largest and smallest oscillating eigenvalue by scaling the spectral radius by 1.05 and 0.05, respectively. Section 5.2 describes how these factors were found.

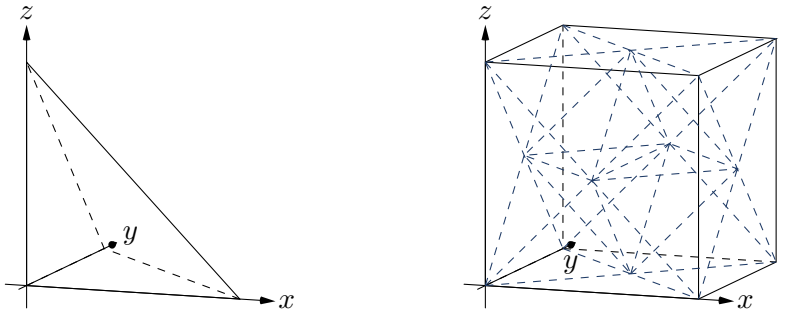
Chapter 5

Results

In this chapter we perform several numerical experiments to investigate the correctness and efficiency of our implementation. All experiments in this thesis can be reproduced on the git commit `c2d063018` which is available at `https://i10git.cs.fau.de/hyteg/hyteg/-/tree/c2d063018a4f217f40141f89272b9452c11d04c2`. The source code is located in the directory `apps/2023-bauer-mt`.

5.1 Domains and Constructed Solutions

The following experiments use as domain the reference tetrahedron (2.49) or the unit cube $\Omega = (0, 1)^3$. The unit cube is subdivided into 24 macro-tetrahedra as shown in Figure 5.1b.



(a) Reference tetrahedron.

(b) Unit cube with 24 macro-elements.

Figure 5.1: Domains and their macro-triangulation.

On each domain we construct two solutions $\boldsymbol{u}^*$ which satisfy homogeneous tangential boundary conditions and from them determine the right-hand side (RHS) $\boldsymbol{f}$. In both cases we design a polynomial function of low order and a function involving sine-waves. The analytical solutions and RHSs read:

- Polynomial on reference tetrahedron:

$$\boldsymbol{u}^* = \begin{pmatrix} yz(x+y+z-1) \\ xz(x+y+z-1) \\ xy(x+y+z-1) \end{pmatrix} \tag{5.1a}$$

$$\boldsymbol{f} = \begin{pmatrix} yz(x+y+z-1) - y - z \\ xz(x+y+z-1) - x - z \\ xy(x+y+z-1) - x - y \end{pmatrix} \tag{5.1b}$$

- Sinusoidal on reference tetrahedron:

$$\boldsymbol{u}^* = \begin{pmatrix} \sin(y)\sin(z)\sin(x+y+z-1) \\ \sin(x)\sin(z)\sin(x+y+z-1) \\ \sin(x)\sin(y)\sin(x+y+z-1) \end{pmatrix} \tag{5.2a}$$

$$\boldsymbol{f} = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \end{pmatrix} \tag{5.2b}$$

$$\begin{aligned} f_1 = &-\sin(x)\sin(y)\sin(x+y+z-1) - \sin(x)\sin(z)\sin(x+y+z-1) + \\ &5\sin(y)\sin(z)\sin(x+y+z-1) + \sin(y)\cos(x)\cos(x+y+z-1) - \\ &2\sin(y)\cos(z)\cos(x+y+z-1) + \sin(z)\cos(x)\cos(x+y+z-1) - \\ &2\sin(z)\cos(y)\cos(x+y+z-1) \end{aligned} \tag{5.2c}$$

$$\begin{aligned} f_2 = &-\sin(x)\sin(y)\sin(x+y+z-1) + 5\sin(x)\sin(z)\sin(x+y+z-1) + \\ &\sin(x)\cos(y)\cos(x+y+z-1) - 2\sin(x)\cos(z)\cos(x+y+z-1) - \\ &\sin(y)\sin(z)\sin(x+y+z-1) - 2\sin(z)\cos(x)\cos(x+y+z-1) + \\ &\sin(z)\cos(y)\cos(x+y+z-1) \end{aligned} \tag{5.2d}$$

$$\begin{aligned} f_3 = &5\sin(x)\sin(y)\sin(x+y+z-1) - \sin(x)\sin(z)\sin(x+y+z-1) - \\ &2\sin(x)\cos(y)\cos(x+y+z-1) + \sin(x)\cos(z)\cos(x+y+z-1) - \\ &\sin(y)\sin(z)\sin(x+y+z-1) - 2\sin(y)\cos(x)\cos(x+y+z-1) + \\ &\sin(y)\cos(z)\cos(x+y+z-1) \end{aligned} \tag{5.2e}$$

- Polynomial on unit cube:

$$\boldsymbol{u}^* = \begin{pmatrix} y(1-y)z(1-z) \\ x(1-x)z(1-z) \\ x(1-x)y(1-y) \end{pmatrix} \tag{5.3a}$$

$$\boldsymbol{f} = \begin{pmatrix} 2(y(1-y)+z(1-z)) + y(1-y)z(1-z) \\ 2(x(1-x)+z(1-z)) + x(1-x)z(1-z) \\ 2(x(1-x)+y(1-y)) + x(1-x)y(1-y) \end{pmatrix} \tag{5.3b}$$

- Sinusoidal on unit cube:

$$\boldsymbol{u}^* = \begin{pmatrix} \sin(2\pi y)\sin(2\pi z) \\ \sin(2\pi x)\sin(2\pi z) \\ \sin(2\pi x)\sin(2\pi y) \end{pmatrix} \tag{5.4a}$$

$$\boldsymbol{f} = \begin{pmatrix} \sin(2\pi y)\sin(2\pi z) + 8\pi^2 \sin(2\pi y)\sin(2\pi z) \\ \sin(2\pi x)\sin(2\pi z) + 8\pi^2 \sin(2\pi x)\sin(2\pi z) \\ \sin(2\pi x)\sin(2\pi y) + 8\pi^2 \sin(2\pi x)\sin(2\pi y) \end{pmatrix} \tag{5.4b}$$

5.2 Eigenvalue Bounds for the Chebyshev Smoother

Our first objective is to find good eigenvalue bounds for the Chebyshev smoother. To that end, we solve (1.2) on the unit cube with $\alpha = \beta = 1$. To eliminate the discretization error, we set the RHS to zero and use a random initial guess. The system is solved by means of 12 multigrid V(1,1)-cycles, i.e. with one pre- and post-smoothing step each. PETSc's conjugate gradient (CG) implementation is used as coarse grid solver on level 0. One iteration of the Gauss-Seidel method is used to smooth in potential space. For smoothing in the Nédélec-space, we perform one iteration of the Chebyshev smoother of order 4. To find eigenvalue bounds, the spectral radius ρ of the scaled system matrix is obtained by performing 40 steps of power iteration. This estimate is scaled by factors l and u yielding the eigenvalue interval $[l\rho, u\rho]$ for the Chebyshev smoother.

We try different factors l and u, and monitor the convergence rate of the solver from the reduction of the 2-norm of the error:

$$\left(\frac{\|\boldsymbol{e}^{(n)}\|_2}{\|\boldsymbol{e}^{(0)}\|_2} \right)^{\frac{1}{n}} . \tag{5.5}$$

The results are given in Table 5.1. Choosing 0.05ρ as lower eigenvalue bound consistently outperforms the other tested options. Both lower and higher values result in slower convergence. In contrast, there does not seem to be a definite best value for the upper bound. Only for the largest tested value (1.2ρ), the convergence factor increases slightly. Based on these results, we select $l = 0.05, u = 1.05$ and determine the coefficients of the Chebyshev polynomials from the interval $[0.05\rho, 1.05\rho]$ in all following experiments.

5.3 Effectiveness of the Hybrid Smoother

The hybrid smoother is evaluated in isolation in Section 3.2.4.

l \ u	1.03	1.05	1.08	1.12	1.2
0.03	0.223	0.226	0.228	0.229	0.244
0.05	0.215	0.214	0.219	0.215	0.226
0.08	0.234	0.232	0.239	0.243	0.254
0.12	0.278	0.287	0.283	0.288	0.300
0.2	0.344	0.353	0.354	0.353	0.364

(a) Level 4.

l \ u	1.03	1.05	1.08	1.12	1.2
0.03	0.282	0.273	0.276	0.278	0.280
0.05	0.271	0.271	0.272	0.270	0.276
0.08	0.277	0.272	0.272	0.281	0.277
0.12	0.299	0.301	0.304	0.311	0.317
0.2	0.358	0.361	0.365	0.368	0.379

(b) Level 5.

Table 5.1: Convergence factor for different lower and upper bound factors l and u.

5.4 Grid-Independent Convergence

Our next test aims at checking that the multigrid V-cycle is an effective solver for the discretized system and that the number of iterations does not deteriorate as the number of unknowns is increased. Most parameters are the same as above but now, three pre- and post-smoothing steps are utilized.

On each level we count the number of V-cycles that are needed to reduce the 2-norm of the residual to 10^{-6} of the initial value. In addition, we compute the factor by which the residual is reduced in a single iteration. Refer to Table 5.2 for the results. First and foremost, it is evident that the multigrid scheme is able to reduce the residual within a couple of iterations. Moreover, we perceive only a mild increase in the convergence factor as the grid spacing is reduced. These results suggest that the multigrid performance does not deteriorate on fine meshes and is consequently suited for the solution of very large problems.

5.5 L^2 Convergence Rate

With this test we ensure that our solver finds the correct solution to the partial differential equation (PDE) (1.2). The multigrid solver uses the same parameters as in the previous section. However, now the discrete system is solved up to a relative

		Polynomial		Sinusoidal	
Level	DoFs	Its.	Conv. factor	Its.	Conv. factor
3	804	4	$1.66 \cdot 10^{-2}$	4	$1.67 \cdot 10^{-2}$
4	5576	5	$3.73 \cdot 10^{-2}$	5	$3.74 \cdot 10^{-2}$
5	41 360	5	$4.73 \cdot 10^{-2}$	5	$4.74 \cdot 10^{-2}$
6	318 240	5	$5.19 \cdot 10^{-2}$	5	$5.21 \cdot 10^{-2}$
7	2 496 064	5	$5.43 \cdot 10^{-2}$	5	$5.47 \cdot 10^{-2}$

(a) Tetrahedron.

		Polynomial		Sinusoidal	
Level	DoFs	Its.	Conv. factor	Its.	Conv. factor
3	15 512	5	$3.99 \cdot 10^{-2}$	5	$3.57 \cdot 10^{-2}$
4	119 344	5	$5.91 \cdot 10^{-2}$	5	$5.99 \cdot 10^{-2}$
5	936 032	6	$7.25 \cdot 10^{-2}$	6	$7.67 \cdot 10^{-2}$
6	7 413 952	6	$7.78 \cdot 10^{-2}$	6	$8.25 \cdot 10^{-2}$
7	59 015 552	6	$8.06 \cdot 10^{-2}$	6	$8.49 \cdot 10^{-2}$

(b) Cube.

Table 5.2: Number of iterations n to reduce the initial 2-norm of the residual to 10^{-6} and residual reduction factor $\left(\frac{\|\boldsymbol{r}^{(n)}\|_2}{\|\boldsymbol{r}^{(0)}\|_2}\right)^{\frac{1}{n}}$.

residual reduction of 10^{-11} to ensure that the system is solved up to discretization error. After finding the approximate solution $\boldsymbol{u}_h$, the L^2-error is approximated by

$$\|\boldsymbol{e}_h\|_{L^2} \approx \sqrt{(\boldsymbol{u}_h - \boldsymbol{u}_h^*)^T M (\boldsymbol{u}_h - \boldsymbol{u}_h^*)}\,, \tag{5.6}$$

where M is the mass matrix, and $\boldsymbol{u}_h^*$ the projection of the analytical solution into $\mathcal{P}_1^- \Lambda^1(\mathcal{T}_h)$. We expect that $\|\boldsymbol{e}_h\|_{L^2}$ reduces quadratically as $h \to 0$ [17, Theorem 5.8, Remark 18]. Table 5.3 presents the numerically obtained error and convergence rates.

Level	Tet poly	Tet sine	Cube poly	Cube sine
4	$1.107 \cdot 10^{-4}$	$1.032 \cdot 10^{-4}$	$1.602 \cdot 10^{-4}$	$4.996 \cdot 10^{-3}$
5	$2.831 \cdot 10^{-5}$	$2.637 \cdot 10^{-5}$	$4.207 \cdot 10^{-5}$	$1.330 \cdot 10^{-3}$
6	$7.121 \cdot 10^{-6}$	$6.629 \cdot 10^{-6}$	$1.087 \cdot 10^{-5}$	$3.428 \cdot 10^{-4}$
7	$1.783 \cdot 10^{-6}$	$1.660 \cdot 10^{-6}$	$2.789 \cdot 10^{-6}$	$8.737 \cdot 10^{-5}$

(a) L^2-error.

Level	Tet poly	Tet sine	Cube poly	Cube sine
5/4	0.256	0.255	0.263	0.266
6/5	0.251	0.251	0.258	0.258
7/6	0.250	0.250	0.257	0.255

(b) Convergence factor.

Table 5.3: L^2-error and derived convergence factors.

It becomes apparent that our solver indeed finds the correct solution of the PDE. The sinusoidal system on the unit cube stands out as it has a larger error than the other test cases. Presumably, this is because the frequency and henceforth the amplitude of higher order terms is larger in this setup. Most importantly, the convergence factor gets very close to the theoretical limit of 0.25 as the level is increased. However, we also find that on the unit cube the convergence rate is worse than on a single tetrahedron.

5.6 Robustness to Choice of Coefficients

Next, the effect of the coefficients on the convergence rate is studied. Since the convergence in the previous test is limited by the discretization error, we return to the setting from Section 5.2: RHS and analytical solution are zero and the initial guess is a random vector. Like above, the convergence rate is determined from the reduction of the 2-norm of the error after 12 V(1,1)-cycles.

From Table 5.4 we learn that the solver is robust to the values of the coefficients α and β. Interestingly, convergence is particularly fast for very large ratios $\frac{\beta}{\alpha}$. Moreover, the convergence rate increases only moderately as the mesh is further refined. Unfortunately,

α \ β	0.01	1	100
0.01	0.186	0.121	0.095
1	0.172	0.173	0.120
100	0.185	0.172	0.177

(a) Level 3.

α \ β	0.01	1	100
0.01	0.216	0.207	0.095
1	0.215	0.214	0.202
100	0.213	0.216	0.216

(b) Level 4.

α \ β	0.01	1	100
0.01	0.274	0.265	0.096
1	0.268	0.277	0.269
100	0.268	0.273	0.272

(c) Level 5.

α \ β	0.01	1	100
0.01	0.303	0.299	0.171
1	0.301	0.302	0.297
100	0.301	0.302	0.301

(d) Level 6.

Table 5.4: Convergence factor $\left(\frac{\|e^{(n)}\|_2}{\|e^{(0)}\|_2}\right)^{\frac{1}{n}}$ determined by doing $n = 12$ V-cycles on the unit cube.

the results are overall slightly worse than reported in [8] for a similar experiment. However, in that work different smoothers and a hexahedral mesh were chosen.

5.7 Non-Convex Domain

Remember that the theoretical discussion of the hybrid smoother was restricted to convex bounded domains in $\mathbb{R}^3$. It is now investigated if the hybrid smoother can also be applied to more complex domains. To that end, we repeat the previous experiment on a toroidal solid with major and minor radius set to 4.0 and 1.6 units, respectively. Considering that only the ratio $\frac{\beta}{\alpha}$ has played a role above, we fix $\alpha = 1$ and vary only β. The solid torus is approximated by 384 macro-cells like sketched in Figure 5.2. This domain does not only have reentrant corners, it is also not simply connected. Nevertheless, it is desirable to solve problems similar to (1.2) on the solid torus, e.g. in nuclear fusion applications.

The results are given in Table 5.5. Although the convergence rate has worsened compared to the simpler domains, the difference is not dramatic. When interpreting these results it must not be neglected that some elements in the toroidal mesh have a worse aspect ratio than those in the cube. On the positive side, the convergence rate is still stable with respect to the ratio of the coefficients.

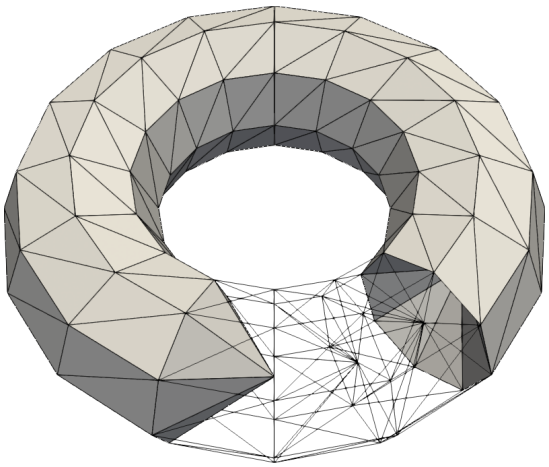


Figure 5.2: Coarse mesh of the toroidal solid consisting of 384 macro-cells.

Level	DoFs	$\beta = 0.01$	$\beta = 1$	$\beta = 100$
2	31 808	0.317	0.302	0.095
3	241 792	0.365	0.358	0.205
4	1 884 416	0.399	0.400	0.332
5	14 877 184	0.423	0.423	0.394

Table 5.5: Convergence factor $\left(\frac{\|\boldsymbol{e}^{(n)}\|_2}{\|\boldsymbol{e}^{(0)}\|_2}\right)^{\frac{1}{n}}$ determined by doing $n = 12$ V-cycles on the toroidal solid.

5.8 Performance

Although performance is not the primary objective of this work, we conduct a simple performance experiment. For this, we solve the model problem with $\alpha = \beta = 1$ on the solid torus by performing 20 V(1,1)-cycles. The coarsest and finest grids are on level 2 and 5, respectively.

The code is executed on two nodes of the Meggie cluster at the Erlangen National High Performance Computing Center (NHR@FAU). A single node is equipped with two Intel Xeon E5-2630v4 "Broadwell" chips each having ten cores and running at 2.2 GHz. All 40 MPI processes hold 52 primitives.

Overall, 12.6 % of the total runtime is spent for estimating eigenvalues while 76.3 % are used for multigrid. The multigrid runtime is split across the levels as follows: level 2: 3.4 %, level 3: 1.9 %, level 4: 11.1 %, level 5: 83.7 %.

Across all levels, the smoother in potential space makes up for 0.6 % of the entire multigrid solve, while the Chebyshev smoother is responsible for 67.4 %. All four grid transfer operators use 3.7 % in total. With 90.1 % the vast majority of the multigrid runtime stems from applying the system matrix. Out of this time 7.4 % are used for communication. This percentage ranges from 6.8 % on level 5 to 24.2 % on level 3.

The main result of this experiment is that the runtimes of the two smoothers differ by factor 119. The fact that the Gauss-Seidel smoother greatly outperforms the Chebyshev scheme is not surprising. While the former simply applies a precomputed stencil, the latter requires four matrix-vector products. Each of these involves calculating the local matrices of every element. Furthermore, care has been taken that the Gauss-Seidel smoother is actually fast to execute. It reportedly achieves approximately 60 % of the theoretical achievable performance according to the roofline model [37]. In contrast, no such efforts have been invested into the Nédélec operator. Even when storing and reusing the element local matrices, the performance difference between the two smoothers is as large as factor 39.

These findings reveal the bottleneck quite clearly: applying the system matrix. Since our coefficients are constant, a stencil based implementation would certainly help to close the performance gap. However, it can not be forgone that in some applications spatially varying coefficients are indispensable. Although the grid transfer operators only have a minor impact in the current setup, compared to the Gauss-Seidel smoother they are actually quite costly. This suggests that after optimizing the smoother for the Nédélec space, the transfer operators, too, require treatment.

Chapter

6 Conclusion and Outlook

In this thesis we implemented a multigrid scheme for problems in $\boldsymbol{H}(\mathbf{curl})$ in the finite element framework *HyTeG*. Specifically, we solved a **curl-curl** model problem which arises from Maxwell's equations. We introduced finite element exterior calculus (FEEC) to learn about structure preserving discretization and the Hodge decomposition. These concepts led to the discretization with Nédélec edge elements and discrete scalar potentials in the form of piecewise Lagrangian polynomials. Moreover, they are the tools required for deriving the hybrid smoother which smoothes both in the Nédélec space and the space of scalar potentials. For the remaining sub-problems, we used Chebyshev and Gauss-Seidel smoothers, respectively.

We implemented linear Nédélec edge elements of first kind and the hybrid smoother in *HyTeG*, using code generation techniques to derive compute kernels of (bi-)linear forms and grid transfer operators automatically. Numerical experiments show that our code exhibits the expected asymptotic convergence rate in the discretization error and is close to grid independent convergence. Moreover, we observed that it is robust with respect to the choice of coefficients and applicable to more complex domains than the theory guarantees.

As much as the author is content with these results, there is room for future enhancement. In Section 5.8 we saw that the overall performance suffers heavily from the elementwise application of the system matrix. An implementation which can make use of the fact that the coefficients are spatially constant is expected to result in a huge performance boost. One difficulty in this regard is that despite the constant coefficients, the element matrices are not constant in vicinity to macro-faces. This is because edges on the boundary of macro-cells are oriented according to the neighboring primitives. If their orientation differs, so do the signs of the basis functions on the boundary and accordingly the element matrices. In addition, the code should be methodically tuned to achieve good node level performance.

After node-level performance optimizations, the communication cost is expected to become significant for large systems. A promising approach is to avoid a large portion of the communication by exploiting linearity of the operators. For example, the sparsity patterns of the grid transfer operators make it possible to skip their

communication entirely within this particular multigrid scheme [38]. It would be very interesting to study how the concepts introduced in [38] carry over to *HyTeG*'s data structures which store data not only on macro-cells but also on the interface primitives of the mesh. Considering that *HyTeG* is a general purpose finite element framework, an implementation should make it hard for users to accidentally skip necessary communication. This is challenging because whether communication is necessary depends both on the overall algorithm and details of the individual operators used therein. Currently, operators are not aware of the larger algorithm they are part of. This prohibits them to make any assumptions about what data is currently available in the ghost layers and thus forces them to update any data they might need. Likewise, the algorithms do not have the necessary information about the operators to know whether communication is required or not.

In this work we focused on linear finite elements. However, going to higher order discretizations can improve the time to solution for a given accuracy. This is because higher order finite element methods (FEMs) are well suited for modern hardware architectures [39]. We identify two major obstacles to increasing the polynomial degree. First, starting at degree 2, Nédélec edge elements incorporate degrees of freedom (DoFs) on the faces of the mesh. The same is true for continuous piecewise Lagrangian elements of order 3 or greater (cf. (2.45)). However, in the three-dimensional case, this is currently not supported by *HyTeG*. Second, implementing each finite element space and its accompanying operators in various order is a significant effort. It is intriguing to study how code generation could be extended to generate not only compute kernels but entire function spaces and operators. Can the generality of the description through differential forms be exploited by the code generator? Is it possible to derive grid transfer operators from their natural mathematical description automatically? These questions pose great starting points for future work.

Chapter 7

Bibliography

[1] R. Wengenmayr, "Funken in der Sternenmaschine," *MaxPlanckForschung*, no. 02/2019, pp. 26–33, 2019. [Online]. Available: https://www.mpg.de/13547528/F002_Fokus_026-033.pdf (visited on Jan. 13, 2023).

[2] ITER Physics Basis Editors, ITER Physics Expert Group Chairs and Co-Chairs, and ITER Joint Central Team and Physics Integration Unit, "Chapter 1: Overview and summary," *Nuclear Fusion*, vol. 39, no. 12, pp. 2137–2174, 1999, Publisher: IOP Publishing. DOI: 10.1088/0029-5515/39/12/301.

[3] U. Trottenberg, C. W. Oosterlee, and A. Schüller, *Multigrid*. San Diego: Academic Press, 2001, ISBN: 978-0-12-701070-0.

[4] D. N. Arnold, *Finite Element Exterior Calculus* (CBMS-NSF Regional Conference Series in Applied Mathematics 93). Philadelphia, PA: Society for Industrial and Applied Mathematics, 2018, ISBN: 978-1-61197-554-3. DOI: 10.1137/1.9781611975543.

[5] C. Pagliantini, "Computational magnetohydrodynamics with discrete differential forms," Ph.D. dissertation, ETH Zürich, 2016. DOI: 10.3929/ETHZ-A-010722079.

[6] A. Galbis and M. Maestre, *Vector Analysis Versus Vector Calculus* (Universitext). Boston, MA: Springer US, 2012, ISBN: 978-1-4614-2200-6. DOI: 10.1007/978-1-4614-2200-6.

[7] J. C. Nedelec, "Mixed finite elements in $\mathbb{R}^3$," *Numerische Mathematik*, vol. 35, no. 3, pp. 315–341, 1980. DOI: 10.1007/BF01396415.

[8] R. Hiptmair, "Multigrid method for Maxwell's equations," *SIAM Journal on Numerical Analysis*, vol. 36, no. 1, pp. 204–225, 1998, Publisher: Society for Industrial and Applied Mathematics. DOI: 10.1137/S0036142997326203.

[9] N. Kohl, D. Thönnes, D. Drzisga, D. Bartuschat, and U. Rüde, "The HyTeG finite-element software framework for scalable multigrid solvers," *International Journal of Parallel, Emergent and Distributed Systems*, vol. 34, no. 5, pp. 477–496, 2019. DOI: 10.1080/17445760.2018.1506453.

[10] C. Großmann and H.-G. Roos, *Numerik partieller Differentialgleichungen* (Teubner-Studienbücher: Mathematik). Stuttgart: Teubner, 1992, ISBN: 3-519-02089-0.

[11] D. Bachman, *A Geometric Approach to Differential Forms*. Boston: Birkhäuser Boston, 2012, ISBN: 978-0-8176-8304-7. DOI: 10.1007/978-0-8176-8304-7.

[12] *Signature (permutation)*, in *Encyclopedia of Mathematics*. [Online]. Available: https://encyclopediaofmath.org/wiki/Signature_(permutation) (visited on Oct. 12, 2022).

[13] D. N. Arnold, R. S. Falk, and R. Winther, "Finite element exterior calculus, homological techniques, and applications," *Acta Numerica*, vol. 15, pp. 1–155, 2006. DOI: 10.1017/S0962492906210018.

[14] *Tangent space*, in *Encyclopedia of Mathematics*. [Online]. Available: https://encyclopediaofmath.org/wiki/Tangent_space (visited on Jan. 16, 2023).

[15] D. N. Arnold, D. Boffi, and F. Bonizzoni, "Finite element differential forms on curvilinear cubic meshes and their approximation properties," *Numerische Mathematik*, vol. 129, no. 1, pp. 1–20, 2015. DOI: 10.1007/s00211-014-0631-3.

[16] M. E. Rognes, R. C. Kirby, and A. Logg, "Efficient assembly of $H(\mathrm{div})$ and $H(\mathrm{curl})$ conforming finite elements," *SIAM Journal on Scientific Computing*, vol. 31, no. 6, pp. 4130–4151, 2010, Publisher: Society for Industrial and Applied Mathematics. DOI: 10.1137/08073901X.

[17] R. Hiptmair, "Finite elements in computational electromagnetism," *Acta Numerica*, vol. 11, pp. 237–339, 2002, Publisher: Cambridge University Press. DOI: 10.1017/S0962492902000041.

[18] W. L. Briggs, V. E. Henson, and S. F. McCormick, *A Multigrid Tutorial, Second Edition*, Second. Society for Industrial and Applied Mathematics, 2000, ISBN: 978-0-89871-950-5. DOI: 10.1137/1.9780898719505.

[19] D. N. Arnold, R. S. Falk, and R. Winther, "Multigrid in $H(\mathrm{div})$ and $H(\mathrm{curl})$," *Numerische Mathematik*, vol. 85, no. 2, pp. 197–217, 2000. DOI: 10.1007/PL00005386.

[20] R. Beck and R. Hiptmair, "Multilevel solution of the time-harmonic Maxwell's equations based on edge elements," *International Journal for Numerical Methods in Engineering*, vol. 45, no. 7, pp. 901–920, 1999. DOI: 10.1002/(SICI)1097-0207(19990710)45:7<901::AID-NME611>3.0.CO;2-4.

[21] D. Braess, *Finite Elemente* (Masterclass), 5th ed. Springer Berlin Heidelberg, 2013, ISBN: 978-3-642-34797-9. DOI: 10.1007/978-3-642-34797-9.

[22] W. Hackbusch, "Multi-grid convergence theory," in *Multigrid Methods*, W. Hackbusch and U. Trottenberg, Eds., ser. Lecture Notes in Mathematics, Berlin, Heidelberg: Springer, 1982, pp. 177–219, ISBN: 978-3-540-39544-7. DOI: 10.1007/BFb0069929.

[23] A. H. Baker, R. D. Falgout, T. V. Kolev, and U. M. Yang, "Multigrid smoothers for ultraparallel computing," *SIAM Journal on Scientific Computing*, vol. 33, no. 5, pp. 2864–2887, 2011, Publisher: Society for Industrial and Applied Mathematics. DOI: 10.1137/100798806.

[24] M. Adams, M. Brezina, J. Hu, and R. Tuminaro, "Parallel multigrid smoothing: Polynomial versus Gauss–Seidel," *Journal of Computational Physics*, vol. 188, no. 2, pp. 593–610, 2003. DOI: 10.1016/S0021-9991(03)00194-3.

[25] O. Axelsson, *Iterative Solution Methods*. Cambridge University Press, 1994, ISBN: 978-0-511-62410-0. DOI: 10.1017/CBO9780511624100.

[26] N. Kohl, D. Thönnes, D. Drzisga, D. Bartuschat, and U. Rüde, "A scalable and modular software architecture for finite elements on hierarchical hybrid grids," in *From Parallel to Emergent Computing*, Andrew Adamatzky, Selim Akl, and Georgios Ch. Sirakoulis, Eds., 1st ed., Boca Raton: CRC Press, 2019, pp. 177–197, ISBN: 978-1-315-16708-4. DOI: 10.1201/9781315167084.

[27] B. Gmeiner, M. Huber, L. John, U. Rüde, and B. Wohlmuth, "A quantitative performance study for stokes solvers at the extreme scale," *Journal of Computational Science*, vol. 17, pp. 509–521, 2016. DOI: 10.1016/j.jocs.2016.06.006.

[28] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith, "Efficient management of parallelism in object-oriented numerical software libraries," in *Modern Software Tools for Scientific Computing*, E. Arge, A. M. Bruaset, and H. P. Langtangen, Eds., Boston, MA: Birkhäuser Boston, 1997, pp. 163–202, ISBN: 978-1-4612-1986-6. DOI: 10.1007/978-1-4612-1986-6_8.

[29] J. Bey, "Tetrahedral grid refinement," *Computing*, vol. 55, no. 4, pp. 355–378, 1995. DOI: 10.1007/BF02238487.

[30] M. E. Go Ong, "Hierachical basis preconditioners for second order elliptic problems in three dimensions," Ph.D. dissertation, Dept. of Applied Mathematics, University of Washington, 1989. DOI: 10.2172/5005434.

[31] D. N. Arnold and A. Logg, "Periodic table of the finite elements," *SIAM News*, vol. 47, no. 9, 2014.

[32] A. Neumaier, *Introduction to numerical analysis*. Cambridge University Press, 2001, ISBN: 978-0-511-61291-6. DOI: 10.1017/CBO9780511612916.

[33] W. Hackbusch, *Theorie und Numerik elliptischer Differentialgleichungen*, 4th ed. Wiesbaden: Springer Fachmedien, 2017, ISBN: 978-3-658-15358-8. DOI: 10.1007/978-3-658-15358-8.

[34] A. Meurer *et al.*, "SymPy: Symbolic computing in Python," *PeerJ Computer Science*, vol. 3:e103, 2017. DOI: 10.7717/peerj-cs.103.

[35] A. Schneebeli, "An $H(\mathrm{curl};\Omega)$-conforming FEM: Nédélec's elements of first type," 2003. [Online]. Available: https://www.dealii.org/reports/nedelec/nedelec.pdf (visited on Jul. 21, 2022).

[36] H. Xiao and Z. Gimbutas, "A numerical algorithm for the construction of efficient quadrature rules in two and higher dimensions," *Computers & Mathematics with Applications*, vol. 59, no. 2, pp. 663–676, 2010. DOI: 10.1016/j.camwa.2009.10.027.

[37] N. Kohl and U. Rüde, "Textbook efficiency: Massively parallel matrix-free multigrid for the stokes system," *SIAM Journal on Scientific Computing*, vol. 44, no. 2, pp. C124–C155, 2022, Publisher: Society for Industrial and Applied Mathematics. DOI: 10.1137/20M1376005.

[38] G. Haase, M. Kuhn, and U. Langer, "Parallel multigrid 3D Maxwell solvers," *Parallel Computing*, vol. 27, no. 6, pp. 761–775, 2001. DOI: 10.1016/S0167-8191(00)00106-X.

[39] P. D. Brubeck and P. E. Farrell, *Multigrid solvers for the de Rham complex with optimal complexity in polynomial degree*, 2022. DOI: 10.48550/arXiv.2211.14284.