Diss. ETH No. 15750

hp-Finite Element Methods on Anisotropically, Locally Refined Meshes in Three Dimensions with Stochastic Data

A dissertation submitted to the
SWISS FEDERAL INSTITUTE OF TECHNOLOGY ZÜRICH

for the degree of
Doktor der Mathematik

presented by
PHILIPP FRAUENFELDER
Dipl. Math. ETH
born September 19, 1974
citizen of Henggart ZH and Niederglatt ZH, Switzerland

accepted on the recommendation of
Prof. Dr. Christoph Schwab, examiner
Prof. Dr. Ralf Hiptmair, co-examiner

2004

Dank

Die vorliegende Doktorarbeit ist während meiner vier-einhalb-jährigen Assistenzzeit am Seminar für Angewandte Mathematik unter der Leitung von Prof. Dr. Christoph Schwab entstanden. Ich bedanke mich bei ihm für die Unterstützung, die Ratschläge und die grosse Freiheit, die ich während meiner Arbeit geniessen durfte.

Bei meinem Korreferenten Prof. Dr. Ralf Hiptmair bedanke ich mich für die konstruktive Kritik an dieser Arbeit und für die Unterstützung während ihrer Entstehung.

Weiter bedanke ich mich bei meinen Bürokollegen Dr. Thomas Wihler, Andreas Rüegg und Patrick Huguenot. Wir haben viel über Mathematik und das Leben diskutiert. Sie haben dazu beigetragen, dass ich mich am Seminar für Angewandte Mathematik gut eingelebt habe und mich in der Gruppe wohl gefühlt habe.

Ein grosser Dank gebührt auch Dr. Christian Lage, Gregor Schmidlin und Kersten Schmidt für die gute Zusammenarbeit und die inspirierenden Diskussionen bei der Weiterentwicklung von Concepts. Prof. Dr. Lezsek F. Demkowicz, Dr. Paul Ledger, Kersten Schmidt, Dr. Andrea Toselli und Radu Todor danke ich für das Korrekturlesen meiner Dissertation.

Ein besonderes Dankeschön gilt meiner Frau Claudia, die mich immer liebevoll unterstützt hat.

Schliesslich gilt mein Dank auch Prof. Dr. Rolf Jeltsch und Prof. Dr. Ralf Hiptmair und allen weiteren Mitarbeiterinnen und Mitarbeitern des SAM für die interessante Zeit, die ich am SAM erlebt habe.

Zürich, im September 2004 Philipp Frauenfelder

Abstract

The present thesis is concerned with hp-Finite Element Methods in three dimensions. To resolve singularities or characteristic length scales in many physical and engineering applications, it is necessary to have *geometric meshes with local refinements* to obtain *exponential convergence*. With exponential convergence, it is feasible to reduce the discretisation error and the numerical errors several orders of magnitude below the modeling error.

In three dimensions, geometric meshes require *simultaneous, anisotropic refinements* in the mesh size h and the polynomial degree p. We study the algorithmic and implementational details and give a number of numerical examples for the reaction diffusion equation as well as Maxwell's equations. Maxwell's equations are discretised using H^1-conforming elements and weighted regularisation.

Uncertainty in the input data is another source of errors besides numerical and modelling errors. We develop a method to solve elliptic partial differential equations with stochastic coefficients efficiently using a Karhunen-Loève expansion of the stochastic coefficients. The numerical scheme is *embarassingly parallel.*

The software used to solve the numerical examples in this thesis is avaible for download as Open Source Software.

Kurzfassung

Die vorliegende Doktorarbeit behandelt hp-Finite Element Methoden in drei Dimensionen. Um Singularitäten oder charakteristische Längen in physikalischen oder technischen Anwendungen aufzulösen und exponentielle Konvergenz zu erhalten, braucht es *geometrische Gitter mit lokalen Verfeinerungen*. Durch die exponentielle Konvergenz wird es möglich, die Diskretisierungs-Fehler und numerischen Fehler um Grössenordnungen unter den Modellierungs-Fehler zu senken.

Um dreidimensionale, geometrische Gitter herstellen zu können, werden *gleichzeitige, anisotrope Verfeinerungen* der Gitterweite h und des Polynomgrads p benötigt. Wir untersuchen die algorithmischen und programmiertechnischen Einzelheiten und geben eine Reihe von Beispielen der Reaktions-Diffusions-Gleichung und der Maxwell-Gleichungen. Die Maxwell-Gleichungen werden mit H^1-konformen Elementen und gewichteter Regularisierung diskretisiert.

Unsicherheit bei den Eingabe-Daten ist eine weitere Quelle von Fehlern neben numerischen und Diskretisierungs-Fehlern. Wir untersuchen eine Methode, um elliptische, partielle Differential-Gleichungen mit stochastischen Koeffizienten effizient zu lösen. Dazu wir eine Karhunen-Loève-Zerlegung der stochastischen Koeffizienten verwendet. Die numerische Methode ist *beschämend parallel*.

Die Software, die für die numerischen Beispiele dieser Arbeit benutzt worden ist, ist als Open Source Software zum Download erhältlich.

Contents

Introduction

Why do we compute?

This thesis is in part concerned with software design for simulation of physical phenomena in the sciences and engineering towards the end of *validation of mathematical models* for these phenomena.

Mathematical Models of Physical Reality

A model is a quantitative and abstract description of physical reality. A *language* (with a consistent grammar in the sense of Noam Chomsky [23]) to formulate models is *mathematics.*

We call a mathematical description of physical reality a *mathematical model M* which ideally accounts for all known information as well as for the uncertainty in our knowledge about the phenomenon to be simulated. Mathematical models continually change and increase in sophistication through *experimental validation.* Validation of a mathematical model of physical reality involves, as a key step, obtaining quantitative and experimentally verifiable predictions from it. This is done today almost exclusively by *numerical simulation*. If the mathematical model is too complex for numerical simulation (as is often the case), simplified *working models* M' which are derived from M and which are mathematically consistent with it are used as basis for numerical simulation.

The two processes, model simplification for simulation and increasing model sophistication due to validation, have led, in many areas of science, to *model hierarchies*, which should be intrinsically consistent and experimentally validated in suitable parameter ranges.

Error Control

In *numerical simulation for the purpose of validation* of mathematical models of physical reality, a necessary prerequisite is the quantitative control of *numerical* and *disretisation errors*, *i. e.* the discretisation has to be verified. Essentially, contributions to simulation errors come from numerical errors (roundoff errors etc.), discretisation errors and modelling errors. We assume here that roundoff errors are controled by using extended precision computations and ignore them.

To separate modelling and discretisation errors, a numerical method must then be designed so as to

- reduce discretisation errors several orders of magnitude below any uncertainty inherent in the model's input data (such as loads, domains, coefficients),
- perform *robustly* over a range of input parameters in the working models that is strictly larger than the range for which the mathematical (working) model is to be validated.

These two issues are addressed in the present dissertation by the systematic use of *high-order* discretisation methods for partial differential equations with particular emphasis on

- *Electromagnetics*, where Maxwell's equations describe physics to a very high degree of precision and where material parameters such as permittivity and dielectric constants are known to a high degree of accuracy,
- *Diffusion problems with stochastic coefficients*, where (elliptic) stochastic partial differential equations are the tool for uncertainty description and quantification in physical systems.

As was shown in the 90ies, high-order hp and spectral methods allow, by judicious combination of local mesh refinement and order adaptation, to exploit analytic regularity of solutions to achieve *robust exponential convergence* of computer models towards mathematical models, thereby practically eliminating discretisation errors. Accordingly, we develop here two such methods: an hp-class of subspaces for elliptic partial differential equations in general polyhedra in $\mathbb{R}^3$ and a spectral approach to the deterministic simulation of elliptic PDEs with stochastic coefficients.

Design Principles for Simulation Software

A key component in numerical simulation is the design of simulation software. Early programming languages like Fortran were effective in translating mathematical formulas of particular discretisations of certain mathematical models into machine executable alpha-numerical operations. They were strongly limited in mapping abstract mathematical objects like subspaces or operators or even symbolic operations bijectively into machine code.

With the massive increase in available CPU and memory per cost, and the advent of object oriented programming languages, the perspective of *mapping mathematical models bijectively to simulation software* became realistic. We believe such bijective mapping of mathematical grammar to simulation software structures to be desirable to guide reusable software development: Mathematical grammar has produced consistent and quantitative abstractions of physical systems valid hundreds of years. Software design, ad-hoc or systematic, has to date led in the best case to codes and grammars that are considerably shorter-lived and which are, as a rule, inconsistent and incompatible with each other.

With the simulation software *Concepts* [26, 54, 73, 74] presented in this dissertation, computer analogs of mathematical concepts are built by object oriented design using the functionalities of *specialisations of abstract classes*[1] in the object oriented language C++ [104, 105].[2] In this way, we transfer universality and consistency of mathematical grammar to simulation software design.[3]

Consider the class of mathematical models (using a so-called *variational formulation*)

$$\mathcal{M} := \{\text{Find } u \in U \text{ such that} \quad a(u, v) = l(v) \quad \forall v \in V\}.$$

[1] An instance (or object) of a *class* is a collection of state (data in the variables of the object) and behaviour (interface of the class). The behaviour is common to all objects of the same class whereas each instance has its own set of variables and therefore a (possibly) different state. An *abstract class* is a class without instances. It has no variables and the interface is only declared but not implemented. An abstract class is used as a parent for derived classes (to prescribe the interface of the children) [21].

[2] It goes without saying that any other object oriented programming language would be equally suitable to achieve this mission. However, C++ is one of the few languages enabling us to achieve Fortran like alpha-numerical performance [101].

[3] Only time can tell if this approach is successful but the grammar of mathematics is, to date, in our opinion the best tool available for this task.

Symbol	Mathematical	Physical
U, V	Linear spaces (function spaces of Sobolev type), finite or infinite dimensional	Finite energy solutions
$a(.,.)$	Bilinear (non-linear) operator	Conservation law and constitutive law
$l(.)$	Linear (non-linear) functional	External forces, sources, sinks

Table 0.1: Fundamental concepts used in the definition of $\mathcal{M}$ from a mathematical and physical point of view.

The concepts used in $\mathcal{M}$ are summarised in Table 0.1. The simulation software should be designed such that all models in $\mathcal{M}$ can be solved.

One way of solving a problem $P \in \mathcal{M}$ is to *discretise* the infinite dimensional function spaces U, V in $\mathcal{M}$ with finite dimensional subspaces $U_N \subset U$ and $V_M \subset V$, thereby obtining a class $\mathcal{D}$ of discretisations of $\mathcal{M}$:

$$\mathcal{D} := \{\text{Find } u_N \in U_N \text{ such that } \quad a_{NM}(u_N, v_M) = l_{NM}(v_M) \quad \forall v_M \in V_M\} \subset \mathcal{M}.$$

$a_{NM}(.,.)$ is the restriction of $a(.,.)$ to $U_N \times V_M$ with additional simplifications like numerical integration (which might introduce errors). U_N is the function space for the approximation of the solution and the proper design of V_M asserts stability of the approximation. It is conceivable that the mathematical model is a-priori finite dimensional without a discretisation (*e. g.* particle models, atomistic simulations)—these models also belong to $\mathcal{D}$. On the other hand, mathematical models belonging to $\mathcal{M} \setminus \mathcal{D}$, *i. e.* infinite dimensional models, could be directly realized on the computer using symbolic simulation.

In this work, we restrict ourselves to models $\mathcal{M}$ which (after discretisation) belong to $\mathcal{D}$. The hierarchic structure of the mathematical models in $\mathcal{M}$ and their instances in $\mathcal{D}$ can be mapped to the software as well. In addition, we enforce the usage of the correct mathematical grammer by the type checking of the C++ compiler.[4] We chose the *inheritance paradigm* in object oriented programming languages to satisfy this. Quoting Timothy A. Budd [21]:

> Inheritance means that the behaviour and data associated with child classes are always an extension of the properties associated with parent

[4] One has to get accustomed to these strict guidelines first when using Concepts. However, it ensures the conservation of the intended structures over a long time and many different users.

classes. On the other hand, since a child class is a more specialised (or restricted) form of the parent class, it is also, in a certain sense, a contraction of the parent class.

This is exactly the same situation we observe for a concrete problem $P \in \mathcal{M}$ and its discretisation $P' \in \mathcal{D} \subset \mathcal{M}$ or the relation of the spaces $U_N \subset U$.

Therefore, we choose the following approach: represent each concept in Table 0.1 by an abstract class and combine them according to the mathematical grammar. This defines *concept oriented design* [74]. Another great advantage of concept oriented design, besides leading to a good design of the classes, is: Mathematicians and other researchers will find the concepts familiar.

Simulation Software *Concepts*

The design ideas and principles above are used in the software *Concepts* [26, 54, 73, 74]. Concepts is a class library written in C++ [104, 105]. We are currently working with Concepts at our institute with *Boundary Element Methods* (BEM) [93], *Finite Element Methods* (FEM) and *generalised FEM* [82]. Below, we outline the main classes of the software by means of a general problem. In addition, an application of the classes to FEM is shown.

Main Classes in Concepts

The discretised problem may be written as a linear system of equations by choosing a suitable basis for each of the subspaces. To keep the discussion focused, assume the same basis $\{\Phi_1, ..., \Phi_N\}$ for both spaces $U_N = V_M$, and obtain

$$\underline{A}^\top \boldsymbol{u} = \boldsymbol{l}$$

with $[\underline{A}]_{ij} := a(\Phi_i, \Phi_j)$ the entries of the *stiffness matrix*[5] $\underline{A}$, $l_i := l(\Phi_i)$ the entries of the *load vector* $\boldsymbol{l}$ and $\boldsymbol{u}$ the coefficient vector of the discrete solution: $u_N = \boldsymbol{\Phi}^\top \boldsymbol{u}$.

The mathematical concepts used above are easily listed: *operator, function, bilinear form, linear form, (sub)space, basis function, matrix, vector* (*c. f.* Figure 0.1—the Unified Modelling Language UML [89] is a graphical language for

[5] We shall always call $\underline{A}$ stiffness matrix and $\boldsymbol{l}$ load vector even though this originates from the application in linear elasticity.

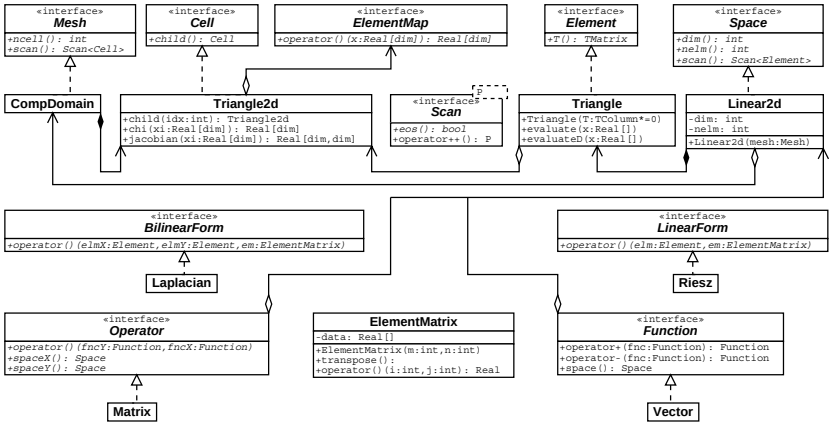


Figure 0.1: UML diagram of the main classes in Concepts together with some specialisations to illustrate the use.

describing object oriented software). Mapping these concepts into classes, however, rises the problem to represent functions, in particular, basis functions. A computationally sensible way to approach this problem is to decompose (mesh) the domain D of the function into primitive sets K_i (*cells* or *elements*). These sets themselves can be characterised by applying mappings F_{K_i} (so-called *element maps*) to predefined reference sets, *e. g.* $F_{K_i} : \hat{K} \to K_i$ such that $K_i = F_{K_i}(\hat{K})$. In addition, functions N_j mounted on a reference set define so-called *shape functions* $\varphi_j^{K_i}$ on each of the elements K_i via $\varphi_j^{K_i} \circ F_{K_i} = N_j$. The basis functions can then be represented by linear combinations of the $\varphi_j^{K_i}$ (extended with 0 to the computational domain D). This generates basis functions $\{\Phi_i\}$ with local support and—as a consequence—a sparse stiffness matrix $\underline{A}$.

Application to FEM: Concrete Classes in Concepts

Consider the scalar model problem given by the *conservation law*

$$f + \nabla \cdot \boldsymbol{F} = 0 \text{ in } D,$$

```
Domain mesh;
Linear2d space(mesh);       // elements are generated

Laplace a;
SparseMatrix A(space, a);   // computing and assembling the stiffness matrix

Riesz f;
Vector l(space, f);         // load vector
Vector u(space);            // empty solution vector

CG solver(A);
solver(l, u);               // solve linear system
```

Algorithm 0.1: An application in pseudo code to solve a diffusion equation in two dimensions.

where f models the sources and forces and $\boldsymbol{F}$ the flux. The *constitutive law* of the *diffusion problem* reads

$$\boldsymbol{F} = \underline{A}\nabla u.$$

$\underline{A}$ is the diffusion tensor and u the concentration. Assuming $\underline{A} = \mathbb{I}$ leads to the Laplace equation

$$-\Delta u = f.$$

Let $V = U$ be the function space of physically admissible concentrations of 'finite energy':
Find $u \in V$ such that

$$a(u, v) = l(v) \quad \forall v \in V,$$

where

$$a(u, v) = \int_D \nabla u \cdot \nabla v \, d\boldsymbol{x}, \qquad l(v) = \int_D f v \, d\boldsymbol{x} + \int_{\Gamma_N} g v \, d\boldsymbol{s}.$$

$g = \boldsymbol{n} \cdot \boldsymbol{F} = \boldsymbol{n} \cdot \underline{A}\nabla u$ on $\Gamma_N \subset \partial D$ is the boundary condition for the flux $\boldsymbol{F}$.

With this background, the necessary classes can be found easily (*c. f.* Figure 0.1). An application in pseudo code looks like in Algorithm 0.1.

Overview of the Thesis

The thesis is subdivided into three parts: *fundamentals, applications* and *software*.

Fundamentals

The first chapter recapitulates the essential tools of the language used to describe the mathematical models: function spaces of Sobolev type, structures (linear and bilinear forms) and abstract projections methods (abstract idea of discretisation).

In the second chapter, the meshes for hp-FEM in two and three dimensions are reviewed following [8, 29] together with the respective convergence results for the FE spaces [8, 97]:

$$\min_{v_N \in V_N} \|u - v_N\|_{H^1(D)} \leq C \exp(-bN^s),$$

where $N = \dim V_N$ and s is $1/3$ and $1/5$ for two and three dimensional problems respectively. Here, u is the solution of a Laplace equation in a Lipschitz polygon or polyhedron.

The first part is closed by the third chapter developing the algorithmic ideas dealing with *hanging nodes.* Hanging nodes arise in locally refined meshes as they are typical for hp-FEM in areas where the mesh is *irregular*. A mesh is called *regular* if the intersection of two different elements is either empty, a vertex or an entire side (edge or face). Irregular meshes do not have this property. The hanging nodes have so-called *constrained degrees of freedom*[6] associated with. These degrees of freedom are constrained by other degrees of freedom to ensure the global continuity of the basis functions of the FE space. There exist different solutions for this *constrained approximation* in the literature and software [14, 15, 41, 42, 43, 47, 50, 68]. We propose a different, very flexible approach [54].

Applications

Maxwell's equations in time harmonic form are treated in the fourth chapter. A variational form for nodal Finite Elements using *weighted regularisation* is reviewed [29]. The regularisation is used to incorporate the divergence condition

$$\operatorname{div} \varepsilon \boldsymbol{E} = 0$$

[6] A degree of freedom corresponds to a global basis function of the FE space.

on the electric field into the variational form:
Find $\boldsymbol{E} \in X_n[Y]$ such that

$$\int_D \mu^{-1} \,\mathbf{curl}\, \boldsymbol{E} \cdot \mathbf{curl}\, \boldsymbol{v}\, d\boldsymbol{x} - \int_D \omega^2 \left(\varepsilon + \frac{\sigma}{i\omega}\right) \boldsymbol{E}\boldsymbol{v}\, d\boldsymbol{x} + \langle \operatorname{div} \boldsymbol{E}, \operatorname{div} \boldsymbol{v}\rangle_Y = - i \int_D \omega \boldsymbol{J}\boldsymbol{v}\, d\boldsymbol{x} \quad \forall \boldsymbol{v} \in X_n[Y].$$

where $X_n[Y] := \{\boldsymbol{u} \in H_0(\mathbf{curl}; D) : \operatorname{div} \boldsymbol{u} \in Y\}$ and $L^2(D) \subset Y \subset H^{-1}(D)$. The numerical experiments to compute the Eigenvalues of Maxwell's equations are conducted on various domains in two and three dimensions [36] yielding exponential convergence with *hp*-FEM.

The second application in Chapter 5 is concerned with *elliptic partial differential equations with stochastic coefficients* [100, 107] and their discretisation with a spectral method in the stochastic variables. This is an attempt to incorporate the uncertainty in the input data into the numerical simulation. A *Karhunen-Loève* [80, 81] *expansion* of the stochastic diffusion coefficient $a(\boldsymbol{x}, \omega)$ in

$$-\operatorname{div}(a(\boldsymbol{x}, \omega)\nabla_x u(\boldsymbol{x}, \omega)) = f(\boldsymbol{x})$$

is used to decouple the stochastic and spatial variables ω and $\boldsymbol{x}$ of the diffusion coefficient a:

$$a(\boldsymbol{x}, \omega) = E_a(\boldsymbol{x}) + \sum_{m \geq 1} \sqrt{\lambda_m}\varphi_m(\boldsymbol{x}) X_m(\omega).$$

$\{(\lambda_m, \varphi_m(\boldsymbol{x}))\}_{m \geq 1}$ are the Eigenpairs of the covariance operator

$$(\mathcal{V}_a u)(\boldsymbol{x}) := \int_D V_a(\boldsymbol{x}, \boldsymbol{x}') u(\boldsymbol{x}')\, d\boldsymbol{x}'$$

associated with the covariance V_a of the diffusion coefficient a. A *Fast Multipole Method for general kernels* [93, 94] is used to compute the Eigenpairs of the covariance operator in a serial computation. *Parallel processing* on a Beowulf type cluster [7, 18] is then used to solve the numerous deterministic FE problems—each with different diffusion and right hand side. Finally, the performance of the Karhunen-Loève expansion Ansatz is compared with a simple *Monte Carlo* method.

Software: Concepts

The third part of this thesis addresses the software Concepts [26, 54, 73, 74] used to compute all numerical examples in the thesis. A substantial part of the research work leading to this thesis was spent in bringing Concepts to its current state.

The first chapter gives detailed information on the realisation of the hp-FE space discretising the Sobolev space $H^1_{\Gamma_D}(D)$ in Concepts. As mentioned earlier, we concentrate on quadrilateral and hexahedral meshes. The chapter explains meshes and their local, anisotropic refinement, the anisotropic handling of the element-wise approximation order p (polynomial degree), the implemented shape functions [72] and the fast integrations techniques used for the element matrices.

The closing Chapter 7 concentrates on the implementation of *vector valued problems* (like Maxwell's equations) and the solution of a so-called *geometric deadlock* problem which is crucial for hp-adaptive mesh refinements in three dimensions.

The framework for vector valued problems implements the notion of a Cartesian product of different FE spaces provided they are all based on the same mesh (there is no restriction on the number of components of the Cartesian product). The given implementation supports block-wise formulation of bilinear and linear forms and reuses as much code as possible from the components of the Cartesian product.

A geometric deadlock originates from refined neighbours of an element with (at first sight) incompatible refinements. The element with the deadlock can therefore not be refined at all. The geometric deadlock problem does not appear in the applications shown in this thesis as only a-priori refinements of the meshes are used. However, with automatic, adaptive refinement, this problem is an issue—but it is solved in Concepts.

Part I

Fundamentals

1 Projection Methods

This chapter reviews the abstract framework and mathematical tools necessary to treat problems of the form
Find $u \in U$ such that

$$a(u, v) = l(v) \quad \forall v \in V.$$

The first section concentrates on general operator equations and their approximate solution by Galerkin projection. Sobolev spaces are introduced in the second section followed by some typical examples of boundary value and Eigenvalue problems in the third section. There is a vast choice of literature on this subject of which only [19, 25, 97] are mentioned.

1.1 Operator Equations

1.1.1 Variational Formulation

We consider a linear *operator equation* with a given $f \in V'$ (the dual space of V): Find $u \in U$ such that

$$Au = f. \tag{1.1}$$

Here, $A : U \to V'$ can be any continuous, linear operator, for instance a differential or integral operator or a combination of these. More specific examples are shown in Section 1.3.

A solution of (1.1) can be found using a *variational formulation.* Define the so-called *bilinear form* $a : U \times V \to \mathbb{R}$ and the *linear form* $l : V \to \mathbb{R}$, *i.e.* $l \in V'$:

$$a(u, v) := \langle Au, v \rangle , \qquad l(v) := \langle f, v \rangle ,$$

where $\langle .\,, . \rangle$ is the duality pairing on $V' \times V$. Then, the variational formulation reads:
Find $u \in U$ such that

$$a(u, v) = l(v) \quad \forall v \in V. \tag{1.2}$$

U and V are called *trial* and *test space* respectively.

1.1.2 Galerkin Projection

In continuum models, U and V are infinite dimensional. To solve (1.2) numerically, a reduction to finite dimension is necessary. One way of doing this consists in restricting (1.2) to finite dimensional subspaces $U_N \subset U$ and $V_M \subset V$. This is called a *discretisation.* Here, $N = \dim U_N$ and $M = \dim V_M$. The finite dimensional problem reads:
Find $u_N \in U_N$ such that

$$a_{NM}(u_N, v_M) = l_{NM}(v_M) \quad \forall v_M \in V_M. \tag{1.3}$$

a_{NM} and l_{NM} are discretisations of a and l respectively (*e. g.* involving numerical integration). Note that (1.3) has the same structure as (1.2).

Choosing bases $\{\Phi_i\}$ of U_N and $\{\Psi_j\}$ of V_M allows to transform (1.3) into a linear system:

$$\begin{aligned} u_N &= \sum u_i \Phi_i = \boldsymbol{u}^\top \boldsymbol{\Phi}, & v_M &= \sum v_j \Psi_j = \boldsymbol{v}^\top \boldsymbol{\Psi}. \\ &\Rightarrow \underline{A}^\top \boldsymbol{u} = \boldsymbol{l}, \end{aligned} \tag{1.4}$$

where

$$[\underline{A}]_{ij} = a(\Phi_i, \Psi_j), \qquad l_j = l(\Psi_j).$$

$\underline{A}$ is called the *stiffness matrix* and $\boldsymbol{l}$ the *load vector*.[1]

From linear algebra, it is well known that (1.4) only has a unique solution, if $N = M$ and $\underline{A}$ is regular.

[1] We shall always call $\underline{A}$ stiffness matrix and $\boldsymbol{l}$ load vector even though this originates from the application in linear elasticity.

Proposition 1.5 (Regular stiffness matrix) *Let* $N = M$. $\underline{A}$ *is regular* $\iff$ $\forall u_N \in U_N \exists v_N \in V_N$ *such that* $a_{NN}(u_N, v_N) \neq 0 \iff \forall v_N \in V_N \exists u_N \in U_N$ *such that* $a_{NN}(u_N, v_N) \neq 0$.

In this case, the bilinear form $a_{NN}(.,.)$ *is called* regular *on* $U_N \times V_N$.

1.1.3 Convergence

The aim of solving (1.3) is to find an approximation u_N of u with a certain *accuracy goal* or *acceptance criterion.* This criterion can be given with respect to a relative error in some norm or a point value, moment or stress of the solution. As long as the accuracy goal is not met, the discretisation is *refined* or *extended* by increasing the dimensions N and M of U_N and V_M respectively. Chapters 2 and 3 give more details on refinements of Finite Element spaces.

Assume $a(.,.) = a_{NM}(.,.)$. Subtracting (1.2) and (1.3) yields

$$a(u - u_N, v_M) = 0 \quad \forall v_M \in V_M \tag{1.6}$$

the so-called *Galerkin orthogonality*: (1.6) states that the discretisation error $u - u_N$ is orthogonal to V_M with respect to the bilinear form $a(.,.)$. If $a(.,.)$ is an inner product of some Hilbert space $H \supset U, V$, (1.6) is the *Galerkin projection* of u onto U_N. We write $u_N = P_N u$. If $a(.,.)$ is regular, the Galerkin projection u_N of u is unique.

Definition 1.7 *A sequence of discretisations* $\{U_{N_i}, V_{M_i}\}_i$ *is*

- stable, *if for all i*
$$\sup_{u \in U} \frac{\left\| P_{N_i} u \right\|_U}{\|u\|_U} \leq K_s < \infty,$$
- consistent, *if for all* $u \in U$
$$\lim_{i \to \infty} \inf_{u_{N_i} \in U_{N_i}} \left\| u - u_{N_i} \right\|_U = 0,$$
- convergent, *if for all* $u \in U$
$$\lim_{i \to \infty} \left\| u - P_{N_i} u \right\|_U = 0.$$

Proposition 1.8 (Lax-Milgram Lemma) *Let $a : U \times V \to \mathbb{R}$ be continuous, i. e.*

$$|a(u, v)| \leq \alpha \|u\|_U \|v\|_V \quad \forall u \in U, v \in V.$$

Let $U_N \subset U$ and $V_N \subset V$ be subspaces of equal dimension N such that the so-called discrete inf-sup condition

$$\inf_{u_N \in U_N} \sup_{v_N \in V_N} \frac{a(u_N, v_N)}{\|u_N\|_U \|v_N\|_V} \geq C_N > 0 \tag{1.9}$$

holds. Then,

- *$a(.,.)$ is regular on $U_N \times V_N$,*
- *the stability constant is $K_s \leq \alpha / C_N$,*
- *the Galerkin projection $P_N u$ is quasioptimal*

$$\|u - P_N u\|_U \leq (1 + \alpha / C_N) \inf_{u_N \in U_N} \|u - u_N\|_U .$$

In the symmetric case, where $a(u, v) = a(v, u)$ and $U = V$, (1.9) is implied by *coercivity:*

$$a(u, u) \geq C \|u\|_U^2 \quad \forall u \in U.$$

1.2 Sobolev Spaces

We briefly review some basics on Sobolev spaces, more information can be found in, *e. g.*, [19, 51, 97].

1.2.1 Definition

Definition 1.10 (Weak partial derivative) *Let $u, v \in L^1_{loc}(D)$, $D \subset \mathbb{R}^d$ and $\boldsymbol{\alpha} \in \mathbb{N}^d$ a multi-index. v is the $\boldsymbol{\alpha}^{\text{th}}$* weak partial derivative *of u:*

$$D^{\boldsymbol{\alpha}} u := \frac{\partial^{\alpha_1}}{\partial x_1^{\alpha_1}} \cdots \frac{\partial^{\alpha_n}}{\partial x_n^{\alpha_n}} u = v$$

if

$$\int_D u D^{\boldsymbol{\alpha}} \varphi \, d\boldsymbol{x} = (-1)^{|\boldsymbol{\alpha}|} \int_D v \varphi \, d\boldsymbol{x} \quad \forall \varphi \in \mathcal{C}_0^\infty(D),$$

where $\mathcal{C}_0^\infty(D)$ *are all compactly supported functions in* $\mathcal{C}^\infty(D)$.

Definition 1.11 (Sobolev space)

$$W^{k,p}(D) := \{u \in L^1_{loc}(D) : D^{\boldsymbol{\alpha}} u \in L^p(D) \quad \forall |\boldsymbol{\alpha}| \leq k\}$$

For $p = 2$, *we write* $H^k(D) = W^{k,2}(D)$. *The* norm *of* $u \in W^{k,p}(D)$ *is defined as*

$$\|u\|_{W^{k,p}(D)} := \begin{cases} \Big(\sum\limits_{|\boldsymbol{\alpha}| \leq k} \int_D |D^{\boldsymbol{\alpha}} u|^p \, d\boldsymbol{x} \Big)^{1/p} & 1 \leq p < \infty \\ \sum\limits_{|\boldsymbol{\alpha}| \leq k} \operatorname*{ess\,sup}_D |D^{\boldsymbol{\alpha}} u| & p = \infty. \end{cases}$$

$W_0^{k,p}(D)$ *denotes the closure of* $\mathcal{C}_0^\infty(D)$ *in* $W^{k,p}(D)$.

Lemma 1.12 $H^k(D)$ *is a Hilbert space with inner product given by*

$$(u, v)_k := \sum_{|\boldsymbol{\alpha}| \leq k} \int_D D^{\boldsymbol{\alpha}} u \cdot D^{\boldsymbol{\alpha}} v \, d\boldsymbol{x}.$$

1.2.2 Traces

To give more flexible boundary conditions than only 0 on the entire boundary of D in the space $W_0^{k,p}(D)$, traces $\gamma_0 u$ of functions $u \in W^{1,p}(D)$ are necessary. There holds [19, 97]:

Proposition 1.13 (Trace theorem) *Let* $D \subset \mathbb{R}^d$ *a bounded domain with Lipschitz boundary* $\Gamma = \partial D$. *The trace operator* $\gamma_0 : u \mapsto u|_\Gamma$ *is continuous from* $W^{1,p}(D)$ *to* $L^p(\Gamma)$ *and there holds*

$$\|\gamma_0 u\|_{L^p(\Gamma)} \leq C \|u\|_{W^{1,p}(D)}, \quad 1 \leq p \leq \infty.$$

C *depends only on* p *and* D.

Proposition 1.13 also holds for parts of the boundary $\Gamma_D \subset \Gamma$ and one can safely define

$$H^1_{\Gamma_D}(D) := \{u \in H^1(D) : u|_{\Gamma_D} = 0\}.$$

Remark 1.14 *$\gamma_0 : H^1(D) \mapsto L^2(\Gamma)$ is not surjective. $\gamma_0(H^1(D))$ maps onto $H^{1/2}(\Gamma)$ with the norm*

$$\|u\|_{H^{1/2}(\Gamma)} := \inf_{\substack{v \in H^1(D), \\ \gamma_0 v = u}} \|v\|_{H^1(D)}.$$

1.3 Examples

This section gives typical examples which fit into the framework of Section 1.1. The first example is the classical reaction diffusion equation (or Laplace or Poisson problem) while the second example treats linear elasticity. Maxwell's equations are shown in the third example. It is detailed in Chapter 4. Further examples are given in Section 2.4.

1.3.1 Reaction Diffusion Equation

Let $D \subset \mathbb{R}^d$, $d = 2$ or 3, be a bounded domain. The reaction diffusion problem reads:

$$-\operatorname{div}(\underline{A}(\boldsymbol{x})\nabla u) + c(\boldsymbol{x})u = f, \tag{1.15}$$

where $\underline{A}$ is the *diffusion matrix*, $c(x)$ the *reaction coefficient* and f is a source term. The diffusion coefficient $\underline{A}(\boldsymbol{x})$ is assumed to be strongly elliptic:

$$c_1|\boldsymbol{\xi}|^2 \leq \boldsymbol{\xi}^\top \underline{A}(\boldsymbol{x})\boldsymbol{\xi} \leq c_2|\boldsymbol{\xi}|^2 \quad \forall \xi \in \mathbb{R}^d,$$

uniformly in $\overline{D}$ and the reaction coefficient $c(\boldsymbol{x}) > 0$. These coefficients are assumed to be sufficiently smooth. This models the *concentration* u of a chemical species in a domain D with sources, reaction and diffusion in steady state (*i. e.* no time dependency). The boundary conditions are

- $u = 0$ on $\Gamma_D \subset \partial D$ (Dirichlet boundary condition),
- $\boldsymbol{n} \cdot \underline{A}(x)\nabla u = g \in H^{-1/2}(\Gamma_N)$ on $\Gamma_N \subset \partial D$ (Neumann boundary condition), $H^{-1/2}(\Gamma)$ is the dual space of $H^{1/2}(\Gamma)$.

Remark 1.16 *To require u on the Dirichlet boundary to be 0 is not a severe restriction since it is always possible to subtract a lifting of a non-homogeneous Dirichlet boundary condition $g_D \in H^{1/2}(\Gamma_D)$ from u and modify the right hand side accordingly. If this is not suitable, the Dirichlet boundary conditions should be handled as constraints [2].*

(1.15) fits into the abstract framework of Section 1.1 with $U = V = H^1_{\Gamma_D}(D)$ and the Hilbert space $H = L^2(D)$:
Find $u \in U = V$ such that

$$\underbrace{\int_D \left(\underline{A}(\boldsymbol{x})\nabla u \cdot \nabla v + c(\boldsymbol{x})uv\right) d\boldsymbol{x}}_{a(u,v)} = \underbrace{\int_D f v \, d\boldsymbol{x} + \int_{\Gamma_N} g v \, d\boldsymbol{s}}_{l(v)} \quad \forall v \in V. \tag{1.17}$$

1.3.2 Linear Elasticity

Let $D \subset \mathbb{R}^d$, $d = 2$ or 3, be a bounded domain. The linear elasticity problem reads [97]:

$$-\operatorname{div} \underline{\sigma}[\boldsymbol{u}] = \boldsymbol{f} \quad \text{in } D, \tag{1.18}$$

with the boundary conditions

$$\boldsymbol{u} = 0 \quad \text{on } \Gamma_D, \qquad\qquad \underline{\sigma}[\boldsymbol{u}] \cdot \boldsymbol{n} = \boldsymbol{g} \quad \text{on } \Gamma_N.$$

Here, $\boldsymbol{u} \in \mathbb{R}^d$ is the *displacement* and $\underline{\sigma}[\boldsymbol{u}]$ the *stress tensor*:

$$\underline{\sigma}[\boldsymbol{u}] := \frac{E}{1+\nu}\underline{\varepsilon}[\boldsymbol{u}] + \frac{\nu}{1-2\nu}\operatorname{div}\boldsymbol{u} \cdot \mathbb{I}, \tag{1.19}$$

where $\underline{\varepsilon}[\boldsymbol{u}]$ is the symmetric gradient of $\boldsymbol{u}$:

$$\left[\underline{\varepsilon}[\boldsymbol{u}]\right]_{ij} = 1/2(\partial_i u_j + \partial_j u_i),$$

$E > 0$ is *Young's modulus* of elasticity and $0 \le \nu \le 1/2$ the so-called *Poisson ratio*.

As $\underline{\varepsilon}[\boldsymbol{u}]$ and $\underline{\sigma}[\boldsymbol{u}]$ are symmetric tensors, they are usually written in the form ($d = 3$)

$$\begin{aligned}
\boldsymbol{\varepsilon} &= \begin{pmatrix} \varepsilon_{11} & \varepsilon_{22} & \varepsilon_{33} & \varepsilon_{12} & \varepsilon_{13} & \varepsilon_{23} \end{pmatrix}^\top, \\
\boldsymbol{\sigma} &= \begin{pmatrix} \sigma_{11} & \sigma_{22} & \sigma_{33} & \sigma_{12} & \sigma_{13} & \sigma_{23} \end{pmatrix}^\top. \\
\boldsymbol{\varepsilon} &= \underline{D}\boldsymbol{u} = \begin{pmatrix} \partial_1 & 0 & 0 \\ 0 & \partial_2 & 0 \\ 0 & 0 & \partial_3 \\ 1/2\partial_2 & 1/2\partial_1 & 0 \\ 1/2\partial_3 & 0 & 1/2\partial_1 \\ 0 & 1/2\partial_3 & 1/2\partial_2 \end{pmatrix} \boldsymbol{u}.
\end{aligned}$$

In addition, (1.19) may be written as $\boldsymbol{\sigma} = \underline{A}\boldsymbol{\varepsilon}$ where $\underline{A} \in \mathbb{R}^{6\times 6}$ is symmetric.

The variational formulation of (1.18) reads:
Find $\boldsymbol{u} \in V = H^1_{\Gamma_D}(D)^d$ such that

$$\underbrace{\int_D \underline{D}\boldsymbol{u} \cdot \underline{AD}\boldsymbol{u}\, d\boldsymbol{x}}_{a(\boldsymbol{u},\boldsymbol{v})} = \underbrace{\int_D \boldsymbol{f}\cdot\boldsymbol{v}\, d\boldsymbol{x} + \int_{\Gamma_N} \boldsymbol{g}\cdot\boldsymbol{v}\, d\boldsymbol{s}}_{l(\boldsymbol{v})} \quad \forall \boldsymbol{v} \in H^1_{\Gamma_D}(D). \tag{1.20}$$

By the first ($|\Gamma_D| > 0$) or second ($|\Gamma_D| = 0$) Korn inequality, $a(.\,,.)$ is coercive. Therefore, (1.20) fits into the setting of Section 1.1 admitting a unique solution with $H = L^2(D)^d$.

1.3.3 Time Harmonic Maxwell's Equations

As before, let $D \subset \mathbb{R}^d$, $d = 2$ or 3, be a bounded domain. The *Maxwell's equations* read [70]:

$$\begin{aligned}
-\partial_t \boldsymbol{D} + \mathbf{curl}\, \boldsymbol{H} &= \sigma \boldsymbol{E} + \boldsymbol{j}^g, & \operatorname{div} \boldsymbol{B} &= 0, \\
\partial_t \boldsymbol{B} + \mathbf{curl}\, \boldsymbol{E} &= 0, & \operatorname{div} \boldsymbol{D} &= \rho,
\end{aligned}$$

where $\boldsymbol{H}$ and $\boldsymbol{E}$ are the *magnetic* and *electric fields* and $\boldsymbol{B}$ and $\boldsymbol{D}$ are the *magnetic* and *electric inductions* respectively. σ is the *conductivity* and $\boldsymbol{j}^g$ the *impressed (given) current density.* The Maxwell's equations model electro-magnetic phenomena. In practice, also problems in unbounded domains are of interest (wave

scattering, antenna modelling etc.) but this is not considered here. For the magnetic and electric inductions, the following constitutive laws hold:

$$\boldsymbol{D} = \underline{\varepsilon}\boldsymbol{E}, \qquad\qquad \boldsymbol{B} = \underline{\mu}\boldsymbol{H}.$$

Hence,

$$\begin{aligned} -\partial_t \underline{\varepsilon}\boldsymbol{E} + \mathbf{curl}\,\boldsymbol{H} &= \sigma\,\boldsymbol{E} + \boldsymbol{j}^g, \\ \partial_t \underline{\mu}\boldsymbol{H} + \mathbf{curl}\,\boldsymbol{E} &= 0 \end{aligned} \tag{1.21}$$

and with the time harmonic Ansatz

$$\begin{aligned} \boldsymbol{E}(\boldsymbol{x},t) &= \mathrm{Re}(\boldsymbol{E}(\boldsymbol{x})\cdot e^{i\omega t}), \qquad & \boldsymbol{H}(\boldsymbol{x},t) &= \mathrm{Re}(\boldsymbol{H}(\boldsymbol{x})\cdot e^{i\omega t}), \\ \boldsymbol{j}^g(\boldsymbol{x},t) &= \mathrm{Re}(\boldsymbol{J}(\boldsymbol{x})\cdot e^{i\omega t}) \end{aligned}$$

and assuming $\underline{\varepsilon}$ and $\underline{\mu}$ independent of t:

$$-\underline{\varepsilon} i\omega \boldsymbol{E} + \mathbf{curl}\,\boldsymbol{H} = \sigma\,\boldsymbol{E} + \boldsymbol{J} \tag{1.22}$$

$$\underline{\mu} i\omega \boldsymbol{H} + \mathbf{curl}\,\boldsymbol{E} = 0. \tag{1.23}$$

Extracting $\boldsymbol{H}$ from (1.23) and inserting it into (1.22) while assuming $\underline{\varepsilon}$ and $\underline{\mu}$ to be scalars (*i. e.* isotropic) formally yields:

$$\mathbf{curl}(\mu^{-1}\,\mathbf{curl}\,\boldsymbol{E}) - \omega^2 \underbrace{\left(\varepsilon + \frac{\sigma}{i\omega}\right)}_{=:\tilde{\varepsilon}} \boldsymbol{E} = -i\omega\boldsymbol{J}, \tag{1.24}$$

the so-called *electric source problem.* Likewise,

$$\mathbf{curl}(\tilde{\varepsilon}^{-1}\,\mathbf{curl}\,\boldsymbol{H}) - \omega^2\mu\boldsymbol{H} = \mathbf{curl}(\tilde{\varepsilon}^{-1}\boldsymbol{J}), \tag{1.25}$$

the so-called *magnetic source problem.*

Assuming no charge density (in the case $\sigma = 0$), *i. e.* $\rho \equiv 0$, implies

$$\mathrm{div}\,\mu\boldsymbol{H} = 0 \text{ and } \mathrm{div}\,\varepsilon\boldsymbol{E} = 0. \tag{1.26}$$

These have to be fulfilled for all solutions of (1.21).

The appropriate space for (1.21) is

$$H(\mathbf{curl};\,D) := \{\boldsymbol{u} \in L^2(D)^d : \mathbf{curl}\,\boldsymbol{u} \in L^2(D)^d\}. \tag{1.27}$$

For a finite electro-magnetic energy, it is required that both fields $\boldsymbol{E}$ and $\boldsymbol{H}$ belong to $H(\mathbf{curl}; D)$. The **curl-curl** forms in (1.24) and (1.25) are not defined for functions in $H(\mathbf{curl}; D)$ and will be dealt with later in the variational formulation.

The boundary conditions for (1.21) of Dirichlet type are

$$\boldsymbol{E} \times \boldsymbol{n} = 0, \qquad \boldsymbol{H} \cdot \boldsymbol{n} = 0 \tag{1.28}$$

and those of Neumann type are

$$\boldsymbol{n} \times (\mu^{-1} \mathbf{curl}\, \boldsymbol{E}) = \boldsymbol{g}, \qquad \boldsymbol{n} \times (\tilde{\varepsilon}^{-1} \mathbf{curl}\, \boldsymbol{H}) = \boldsymbol{h}.$$

The homogeneous Dirichlet boundary conditions (1.28) are the so-called *perfect conductor boundary conditions* on the boundary of the domain D.

Remark 1.29 *In two dimensions, the* **curl** *operator can be defined for a vector field and for a scalar field:* $\operatorname{curl} \boldsymbol{E} = \partial_1 E_2 - \partial_2 E_1$ *(result is a scalar field) and* $\mathbf{curl}\, \varphi = (\partial_2 \varphi, -\partial_1 \varphi)^\top$ *(result is a vector field). The variational space* (1.27) *is defined as*

$$H(\mathbf{curl}; D) := \{\boldsymbol{u} \in \mathcal{D}'(D)^2 : \boldsymbol{u} \in L^2(D)^2, \operatorname{curl} \boldsymbol{u} \in L^2(D)\}.$$

Variational Formulation

Starting from (1.24) and (1.26) (and assuming ε constant),

$$\mathbf{curl}(\mu^{-1} \mathbf{curl}\, \boldsymbol{E}) - \omega^2 \tilde{\varepsilon} \boldsymbol{E} = -i\omega \boldsymbol{J}, \tag{1.30}$$

$$\operatorname{div} \varepsilon \boldsymbol{E} = 0, \tag{1.31}$$

the 'minimal' choice for the variational space would be

$$\{\boldsymbol{u} \in H(\mathbf{curl}; D) : \boldsymbol{u} \times \boldsymbol{n} = 0 \text{ on } \partial D \text{ and } \operatorname{div} \varepsilon \boldsymbol{u} = 0\}.$$

A conforming discretisation would then impose the use of divergence-free elements.

A 'maximal' and more widely used choice for the electric variational space is

$$H_0(\mathbf{curl}; D) := \{\boldsymbol{u} \in H(\mathbf{curl}; D) : \boldsymbol{u} \times \boldsymbol{n} = 0 \text{ on } \partial D\}. \tag{1.32}$$

The corresponding electric variational formulation for (1.30) and (1.31) is then: Find $\boldsymbol{E} \in H_0(\mathbf{curl}; D)$ such that

$$\int_D \left(\mu^{-1}\,\mathbf{curl}\,\boldsymbol{E}\cdot\mathbf{curl}\,\boldsymbol{v} - \omega^2\tilde{\varepsilon}\,\boldsymbol{E}\cdot\boldsymbol{v}\right) d\boldsymbol{x} = \int_D \boldsymbol{f}\cdot\boldsymbol{v}\,d\boldsymbol{x} \quad \forall \boldsymbol{v} \in H_0(\mathbf{curl}; D), \tag{1.33}$$

by using

$$\int_D \mathbf{curl}\,\boldsymbol{u}\cdot\boldsymbol{v}\,d\boldsymbol{x} = \int_D \boldsymbol{u}\cdot\mathbf{curl}\,\boldsymbol{v}\,d\boldsymbol{x} + \int_{\partial D} \boldsymbol{n}\times\boldsymbol{u}\cdot\boldsymbol{v}\,d\boldsymbol{s}$$

where $\boldsymbol{f} = -i\omega\boldsymbol{J}$. The associated bilinear operator is not elliptic and the divergence condition (1.31) is an independent constraint for $\omega = 0$.

Maxwell's equations are discussed in more detail in Chapter 4.

1.3.4 Eigenvalue Problem

Eigenvalue problems need a different setting and give different results (most notably: more than one solution). Anyway, projection methods can also be applied. In this section, the basic facts on Eigenvalue problems are reviewed. These facts, their generalisations and more results can be found in, *e. g.*, [9].

Variational Formulation

Find Eigenpairs $\lambda \in \mathbb{R}$ and $0 \neq u \in V$ ($V \subset H$ a Hilbert space) such that

$$a(u, v) = \lambda b(u, v) \quad \forall v \in V, \tag{1.34}$$

where $a(.\,,.)$ is a symmetric, coercive bilinear form and $b(.\,,.)$ is a symmetric positive definite bilinear form. An Eigenvector of (1.34) is not unique and therefore $\|u\|_H = 1$ is required. If $a(.\,,.)$ is not symmetric, complex Eigenvalues occur.

Assume $V \subset H$ is a compact embedding. There exists a unique bounded operator T satisfying $a(Tu, v) = b(u, v)$. T is self-adjoint and compact. The Eigenpairs (λ, u) from (1.34) fulfil $\lambda Tu = u$ and vice-versa. Therefore, (λ^{-1}, u) is an Eigenpair of T.

The spectrum of a self-adjoint compact operator is bounded and only clusters at 0. Therefore, the Eigenvalues of (1.34) only cluster at ∞ and are bounded from

below: $0 < \lambda_1 \leq \lambda_2 \leq \cdots \infty$. In addition, the λ_j can be characterised by the Rayleigh quotient

$$R(u) := \frac{a(u,u)}{b(u,u)} \tag{1.35}$$

and the Minimum principle:

$$\begin{aligned}\lambda_1 &= \min_{u \in V} R(u) = R(u_1)\\ \lambda_k &= \min_{\substack{u \in V, a(u,u_i)=0\\ \text{for } i=1,\ldots,k-1}} R(u) = R(u_k) \quad k = 2, 3, \ldots.\end{aligned}$$

Choosing a finite dimensional subspace $V_N \subset V$ in (1.34) yields a finite number of Eigenvalues $0 < \lambda_{1,N} \leq \lambda_{2,N} \leq \cdots \leq \lambda_{N,N}$ and corresponding Eigenvectors $u_{1,N}, u_{2,N}, \ldots, u_{N,N}$ satisfying

$$a(u_{i,N}, v_N) = \lambda_{i,N}\, b(u_{i,N}, v_N) \quad \forall v_N \in V_N. \tag{1.36}$$

Approximation Estimates

It follows directly from the Minimum principle that $\lambda_k \leq \lambda_{k,N}$.

Let

$$\begin{aligned}M(\lambda) &:= \{u : u \text{ an Eigenvector corresponding to } \lambda, \|u\|_H = 1\},\\ \varepsilon_N(\lambda) &:= \sup_{u \in M(\lambda)} \inf_{v_N \in V_N} \|u - v_N\|_H.\end{aligned}$$

Proposition 1.37 *Using* (1.36) *with a finite dimensional subspace $V_N \subset V$ yields*

$$\|u_i - u_{i,N}\|_H \leq C\varepsilon_N, \qquad |\lambda_i - \lambda_{i,N}| \leq C\varepsilon_N^2,$$

i. e. the Eigenvalues converge twice as fast as the Eigenvectors. For single Eigenvalues λ_i: $C_1\varepsilon_N^2 \leq |\lambda_i - \lambda_{i,N}| \leq C_2\varepsilon_N^2$.

Remark 1.38 *Generalisations of Proposition 1.37 to the case of multiple Eigenvalues can be found in [9].*

Heat Equation

As before, let $D \subset \mathbb{R}^d$, $d = 2$ or 3, be a bounded domain. The heat equation reads

$$\partial_t u(\boldsymbol{x}, t) - \operatorname{div}(\underline{A}(\boldsymbol{x}, t)\nabla u(\boldsymbol{x}, t)) = f(\boldsymbol{x}, t), \tag{1.39}$$

with a sufficiently smooth coefficient matrix $\underline{A}$ and homogeneous Dirichlet and Neumann boundary conditions. $\underline{A}$ is the *heat conduction coefficient* matrix and f models an *external heating.*

(1.39) should be reformulated as an Eigenvalue problem to look for Eigenmodes of the heat equation. The Ansatz $u(\boldsymbol{x}, t) = v(\boldsymbol{x}) \cdot w(t)$ (so-called separation of variables) while dropping f and assuming $\underline{A}$ and c constant in time leads to

$$v(\boldsymbol{x})\partial_t w(t) - w(t)\operatorname{div}(\underline{A}\nabla v(\boldsymbol{x})) = 0.$$

Assuming v and $w \neq 0$ allows

$$\underbrace{-\frac{\partial_t w}{w}(t)}_{\text{constant in } \boldsymbol{x}} = \underbrace{-\frac{\operatorname{div}(\underline{A}\nabla v)}{v}(\boldsymbol{x})}_{\text{constant in } t} = \lambda.$$

The right hand equation reads

$$\lambda v = -\operatorname{div}(\underline{A}\nabla v).$$

The bilinear forms

$$a(u, v) = \int_D \underline{A}(\boldsymbol{x})\nabla u \cdot \nabla v \, d\boldsymbol{x}$$
$$b(u, v) = \int_D uv \, d\boldsymbol{x}$$

show that (1.39) fits into the framework of the Eigenvalue problem (1.34) with $V = H^1_{\Gamma_D}(D)$ and the Hilbert space $H = L^2(D)$.

2 Finite Element Methods

The Galerkin projection reviewed in the previous chapter is based on finite dimensional subspaces of the test and trial spaces U, V. One way to build such subspaces is the so-called Finite Element Method (FEM).

The first section of this chapter reviews FE meshes and classical approximation results. The second and third sections review more advanced geometric meshes used in hp-FEM in two and three dimensions respectively. The chapter closes with a brief outlook on other Finite Element Methods.

2.1 Finite Element Meshes

The basis functions of the finite dimensional subspaces $U_N \subset U$ and $V_M \subset V$ mentioned in the previous chapter are constructed combining shape functions defined on small subdomains K of the domain of interest D.[1] This is explained in some more detail in Section 2.1.2.

2.1.1 Definition

Following [63], this section gives the definitions for FE meshes.

Definition 2.1 (Cells and mesh) *A* cell K *is an open subdomain of D with piecewise smooth, Lipschitz boundary.*

A mesh $\mathcal{T}$ *is a partition of a bounded domain of interest $D \subset \mathbb{R}^d$ into a finite number of (disjoint) cells K. The collection of cells $\{K\} = \mathcal{T} = \mathcal{T}(D)$, $\overline{D} = \bigcup_{K \in \mathcal{T}} \overline{K}$.*

[1] Most of the time, $N = M$, as a square stiffness matrix is required.

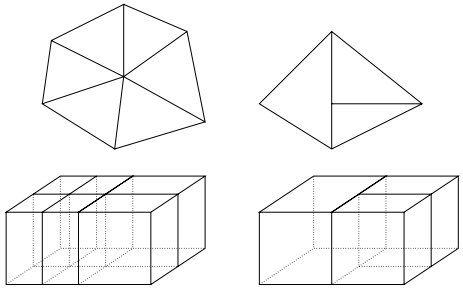


Figure 2.1: Regular meshes on the left and irregular meshes on the right hand side.

In a regular mesh, *the intersection of any two cells* $\overline{K}_i$ *and* $\overline{K}_j$, $i \neq j$, *is either empty, a vertex or an entire side*[2].

In two dimensions, the cells are typically triangles or quadrilaterals. In three dimensions, tetrahedra, hexahedra, prisms and pyramids are usually used. Examples are shown in Figure 2.1.

Remark 2.2 *Normally, FE meshes are required to be regular. However, it is possible to weaken this requirement while still retaining the global continuity of the basis functions of the FE space (c. f. Chapter 3). In this chapter, we assume that the meshes are regular.*

Definition 2.3 (Element) *Each cell* K *of the mesh* $\mathcal{T}$ *is assigned a* polynomial degree[3] $p_K \in \mathbb{N}_0$. *The cell-wise polynomial degrees* p_K *constitute the* degree vector $\boldsymbol{p}$.

Let $\mathcal{P}$ *be a finite dimensional space of functions (the* shape functions*) on the cell* K *and* $\mathcal{N}$ *a basis of* $\mathcal{P}'$ *(the set of* nodal variables*). Then,* $(K, \mathcal{P}, \mathcal{N})$ *is an* element *[19, 24].*

As a short form of $(K, \mathcal{P}, \mathcal{N})$, we write (K, p_K) where $\mathcal{P} = \mathcal{V}_{p_K}$ (the space of polynomials with degree p_K, *c. f.* Remark 2.11 below) and $\mathcal{N}$ the standard nodal variables for $\mathcal{P}$ [19].

[2] A side in two dimensions is an edge and in three dimensions, it is a face (*i. e.* a triangle or a quad).

[3] Usually interpreted as degree of the local space of polynomials.

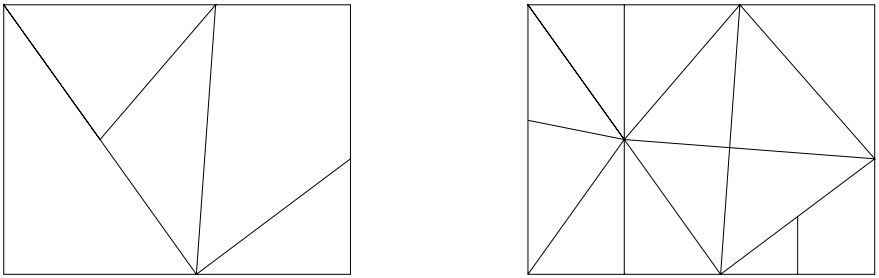


Figure 2.2: An irregular mesh $\mathcal{T}$ (left) and a possible (irregular) refinement (right).

Definition 2.4 (Extension / refinement) *A* subdivision *of a cell K is any partition of K into a finite number of cells K_i, $i \in \mathcal{I}_K$. K_i is called a* child *of K. A mesh-degree combination $(\mathcal{T}, \boldsymbol{p})$ is* extended *or* refined *by:*

- *A subdivision of some or all cells K in $\mathcal{T}$.* *h-extension*[4]
- *An increase of some or all polynomial degrees p_K.* *p-extension*[5]
- *A combination of both.* *hp-extension*[5]

A uniform *extension refines all elements (K, p_K) in the same way (in h and / or p).*

Remark 2.5 *When subdividing a cell K, the associated degree p_K is typically inherited by the children K_i of K.*

An extension (or a refinement) without further information may be regarded as either h-extension, p-extension or hp-extension.

Example 2.6 *Figure 2.2 shows a mesh $\mathcal{T}$ and a possible refinement. Note that both meshes are* irregular.

[4]The diameter of a cell K is usually denoted by h. The h-version is decreasing the maximal h in the mesh $\mathcal{T}$.

[5]In p- and hp-extensions, the maximal diameter of a cell in general does not tend to zero during successive refinements.

2.1.2 Finite Element Subspaces

Using a mesh $\mathcal{T} = \{K\}$ and a degree vector $\boldsymbol{p}$, the Finite Element space V_N is usually constructed as follows.

Each cell K in the mesh has a *master (reference) cell* $\hat{K}$. In two dimensions, triangles K typically are assigned the triangle $\hat{K} = \{\boldsymbol{\xi} \in \mathbb{R}^2 : \xi_i > 0, \xi_1 + \xi_2 < 1\}$ as reference cell and quadrilaterals K are typically assigned one of the reference cells $\hat{K} = (0,1)^2$ or $\hat{K} = (-1,1)^2$. For tetrahedra, the typical reference cell is again the unit simplex and for hexahedra either $\hat{K} = (0,1)^3$ or $\hat{K} = (-1,1)^3$ is used. The choice of the reference cell also fixes the *element map* $F_K : \hat{K} \to K$.

Remark 2.7 *Contrary to the examples of cells in Figures 2.2 and 2.3, also curved boundaries are allowed. In such a case e. g. isoparametric or exact geometry (using blending maps) representation is used. However, this is not covered in this work.*

For cells with straight edges, an affine (triangle and tetrahedron), bilinear (quadrilateral) or trilinear (hexahedron) element map is sufficient.

In addition to the reference cell and the element map, *reference shape functions* N_i of degree p_K are chosen on the reference cell. These reference shape functions N_i are mapped onto the physical cell using the element map: $\varphi_i^K \circ F_K = N_i$. The φ_i are called the (physical) *shape functions.*

Example 2.8 *Figure 2.3 shows an element map F_K for a quadrilateral cell K together with an example for a reference shape function N_i and the respective shape function φ_i on the physical cell.*

The *global basis functions* Φ_i of the FE space V_N are constructed by combining the shape functions on adjacent elements, the so-called *assembly* of global basis functions. How this is achieved depends on the continuity requirements of the FE space. Typically, one requires global $\mathcal{C}^0$ continuity of the basis functions. A rather general method to assemble the global basis functions will be shown in Chapter 3.

Definition 2.9 (FE space) *On a mesh $\mathcal{T} = \{K\}$, define*

$$\mathcal{S}^{\boldsymbol{p},l}(D,\mathcal{T}) := \{u \in H^l(D) : u|_K \circ F_K \in \mathcal{V}_{p_K}(\hat{K}), K \in \mathcal{T}\}, \tag{2.10}$$

i. e. piecewise polynomials of degree p_K on the reference element $\hat{K}$.

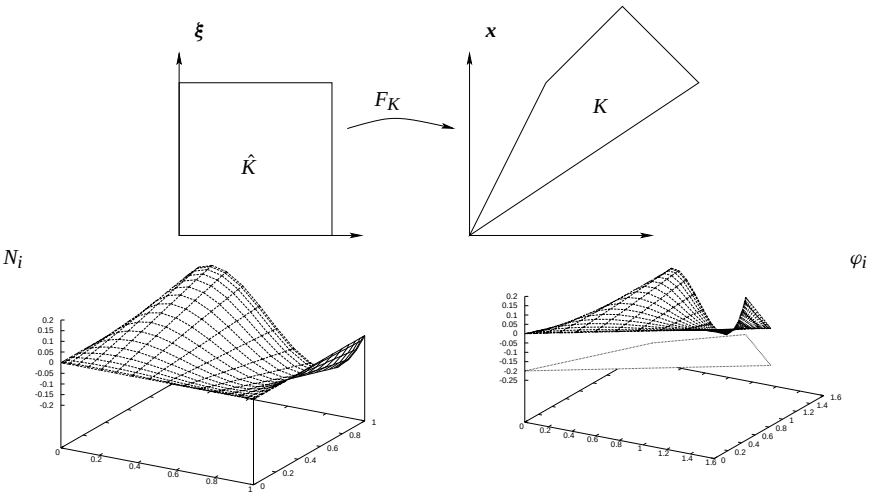


Figure 2.3: The top figure shows the element map $F_K : \hat{K} \to K$. Below, there is a reference shape function N_i on the left and the corresponding shape function φ_i on the physical cell on the right.

Remark 2.11 *The polynomial space $\mathcal{V}_p$ in Definition 2.9 denotes:*

- *The space of polynomials with maximal degree p in* quadrilaterals *and* hexahedra*: $\mathcal{V}_p = \mathcal{Q}_p := \text{span}\{\boldsymbol{x}^{\boldsymbol{\alpha}} : \alpha_i \leq p\}$.*

- *The space of polynomials with total degree p in* simplices *(triangles and tetrahedra): $\mathcal{V}_p = \mathcal{P}_p := \text{span}\{\boldsymbol{x}^{\boldsymbol{\alpha}} : |\boldsymbol{\alpha}| \leq p\}$.*

In classical FEM, $l = 1$. There are other possible choices, though. Refining $\mathcal{S}^{\boldsymbol{p},l}(D, \mathcal{T})$ by p-extension is called p-FEM[6]. Refining $\mathcal{S}^{\boldsymbol{p},l}(D, \mathcal{T})$ by h-extension is called h-FEM[7]. A combination of h- and p-FEM (hp-FEM) is introduced in Sections 2.2 and 2.3.

[6] In p-FEM, the mesh $\mathcal{T}$ is usually regular.

[7] In h-FEM, the degree p is usually uniform and low, *i. e.* $p = 1$ or 2.

2.1.3 Approximation Properties

This section gives approximation properties for h- and p-extensions on simple meshes. A thorough discussion and more results can be found in [25, 97]. More complicated meshes are discussed in Sections 2.2 (two dimensions) and 2.3 (three dimensions).

Definition 2.12 (Mesh properties) *Let $\{\mathcal{T}\}$ be a family of meshes with cells K. In a cell K denote with*

$$\begin{aligned} \rho_K &\text{ radius of the maximal inscribed circle of } K, \\ h_K &\text{ the radius of the circumference of } K. \end{aligned}$$

- $\{\mathcal{T}\}$ *is called* $\varkappa$-shape-regular, *if*

$$0 < \varkappa \leq \min\{h_K/\rho_K : K \in \mathcal{T}\} \leq \max\{h_K/\rho_K : K \in \mathcal{T}\} \leq \varkappa^{-1} < \infty.$$

- $h(\mathcal{T}) := \max\{h_K : K \in \mathcal{T}\}$ *is the* mesh width *of* $\mathcal{T}$.

- $\{\mathcal{T}\}$ *is called* $\tilde{\varkappa}$-quasi-uniform *if*

$$0 < \tilde{\varkappa} \leq \frac{\max_{K\in\mathcal{T}} h_K}{\min_{K\in\mathcal{T}} h_K} \leq \tilde{\varkappa}^{-1} < \infty.$$

Proposition 2.13 (Approximation property of h- and p-FEM)
Let $\mathcal{T}$ be a shape-regular and quasi-uniform mesh with mesh width h in $D \subset \mathbb{R}^d$ a polygon ($d = 2$) or a polyhedron ($d = 3$).

Then (see e. g. [10] and the references there), for a uniform polynomial degree p,

$$\min_{v\in\mathcal{S}^{p,1}(D,\mathcal{T})} \|u - v\|_{H^1(D)} \leq C(k)\,(h/p)^{\min\{p+1,k\}-1}\,\|u\|_{H^k(D)} \quad \forall u \in H^k(D).$$

Combining Proposition 2.13 with Proposition 1.8 or 1.37 gives the usual convergence results for sufficiently smooth solutions.

2.2 Geometric Meshes in Two Dimensions

Proposition 2.13 shows the approximation properties for h- and p-extensions. However, piecewise analytic solutions with a singular behaviour near corners yield only algebraic convergence with h- and p-extensions. The solution is to combine h- and p-extensions in a suitable way to regain the exponential convergence [8, 97].

Remark 2.14 (Geometric meshes necessary for exponential convergence) *Given smooth data, the solution of a reaction diffusion equation*[8] *is analytic in* $\overline{D} \setminus \mathcal{C}$ *(*$\mathcal{C}$ *the set of corners of the domain D) [8, 97]. The singularities spoiling the convergence of the p-FEM only occur at corners in* $\mathcal{C}$*. In any cell* $\overline{K} \subset \overline{D} \setminus \mathcal{C}$*, the solution can be approximated by polynomials of degree p at an exponential rate in p as long as the ratio*

$$\frac{\inf_{x \in K} \operatorname{dist}(x, \mathcal{C})}{\operatorname{diam} K}$$

can be bounded from above and below. In geometric meshes as described below (and constructed in Chapter 3), this is fulfilled.

The description of these geometric hp-meshes follows [8, 29]. How the meshes are realized algorithmically is shown in Chapter 3. The three dimensional case is treated in Section 2.3.

2.2.1 Domains

Let $D \subset \mathbb{R}^2$ be a bounded Lipschitz polygon defined as follows. The definition also includes domains in $\mathbb{S}^2$ as this is needed later for the definition of Lipschitz polyhedra in three dimensions.

Definition 2.15 (Lipschitz polygon in two dimensions) *Let the domain* $D \subset \mathbb{R}^2$ *or* $\mathbb{S}^2$ *be bounded. At each point* $\boldsymbol{a} \in \partial D$*, there exists* $r_{\boldsymbol{a}} > 0$ *and a diffeomorphism* $\chi_{\boldsymbol{a}}$ *such that the neighbourhood* $\mathcal{V}_{\boldsymbol{a}} := D \cap B_{r_{\boldsymbol{a}}}(\boldsymbol{a})$ *of* $\boldsymbol{a}$ *is transformed into a neighbourhood of the corner 0 of a plane sector* $\Gamma_{\boldsymbol{a}}$ *of* opening angle $\omega_{\boldsymbol{a}} \in (0, 2\pi)$ *with* $\boldsymbol{a}$ *being sent into 0. Assume that the diffeomorphism* $\chi_{\boldsymbol{a}}$ *is an isometric transformation at* $\boldsymbol{a}$*. Therefore, the opening angle* $\omega_{\boldsymbol{a}}$ *is an intrinsic parameter of D.*

[8] This does not only hold for reaction diffusion equations but also for other elliptic PDEs (like linear elasticity). However, each problem has to be treated separately.

The set $\mathcal{C}$ of corners *of D consists of points $\boldsymbol{a} \in \partial D$ where*

- *the corresponding sector $\Gamma_{\boldsymbol{a}}$ is non-trivial (i. e. $\omega_{\boldsymbol{a}} \neq \pi$),*
- *the type of boundary conditions change.*

Definition 2.15 splits D into the neighbourhoods $\mathcal{V}_{\boldsymbol{a}}$ for all corners in $\boldsymbol{a} \in \mathcal{C}$ and the *regular part* $\mathcal{V}^0$ such that no vertex $\boldsymbol{a}$ belongs to $\overline{\mathcal{V}^0}$ (all unions are disjoint):

$$D = \mathcal{V}^0 \cup \Big(\bigcup_{\boldsymbol{a} \in \mathcal{C}} \mathcal{V}_{\boldsymbol{a}} \Big).$$

2.2.2 Assumptions on Cells

Consider quadrilateral cells K in a mesh $\mathcal{T} = \{K\}$ covering the whole domain D. F_K denotes the bijective map from the *reference cell* $\hat{K} = (-1, 1)^2$ onto K: $F_K : \boldsymbol{\xi} \mapsto \boldsymbol{x} = F_K(\boldsymbol{\xi})$. Let $h(K) > 0$ be a measure for the size of the element and assume:

1. For any $|\boldsymbol{\alpha}| = m > 0$ and any $\boldsymbol{\xi} \in \hat{K}$, there holds

$$|D^{\boldsymbol{\alpha}}(F_K)_i(\boldsymbol{\xi})| \leq C_0 d_0^m (m!) h(K), \tag{2.16}$$

   for $m > 1$ and $i = 1, 2$.

2. The Jacobian determinant is positive and satisfies

$$C_1 h^2(K) \leq \det \frac{dF_K}{d\boldsymbol{\xi}} \leq C_2 h^2(K). \tag{2.17}$$

Here, the constants C_0, C_1, C_2 and d_0 are the same for all cells K.

Remark 2.18 *The above conditions guarantee the usual assumptions as the angle condition and a bounded aspect ratio of the cells K etc.*

The cells K in the regular part $\mathcal{V}^0$ and the corner neighbourhoods $\mathcal{V}_{\boldsymbol{a}}$ are considered separately.

Cells in a Corner Neighbourhood $\mathcal{V}_{\boldsymbol{a}}$

Let $K \subset \mathcal{V}_{\boldsymbol{a}}$ and $\boldsymbol{a} \notin \overline{K}$. Denote with $r_{\boldsymbol{a}}(\boldsymbol{x}) = |\boldsymbol{a} - \boldsymbol{x}|$ the distance from the corner and

$$\underline{\varkappa}(K) = \min_{\boldsymbol{x} \in \overline{K}} r_{\boldsymbol{a}}(\boldsymbol{x}), \qquad \overline{\varkappa}(K) = \max_{\boldsymbol{x} \in \overline{K}} r_{\boldsymbol{a}}(\boldsymbol{x}). \tag{2.19}$$

Assume constants $A \in (1, \infty)$, $\underline{C}$ and $\overline{C}$ exist such that

$$\frac{\overline{\varkappa}(K)}{\underline{\varkappa}(K)} \leq A \tag{2.20}$$

and

$$\underline{C}\, \underline{\varkappa}(K) \leq h(K) \leq \overline{C}\, \overline{\varkappa}(K). \tag{2.21}$$

There exists

$$1 < A^\star \leq \frac{\overline{\varkappa}(K)}{\underline{\varkappa}(K)}. \tag{2.22}$$

Denoting by $\|K\|$ the area measure of K, there holds

$$\underline{C}\, \underline{\varkappa}^2(K) \leq \|K\| \leq \overline{C}\, \overline{\varkappa}^2(K). \tag{2.23}$$

For the cells $K_{\boldsymbol{a}}$ containing the corner $\boldsymbol{a}$: $\boldsymbol{a} \in \overline{K_{\boldsymbol{a}}}$, assume (2.21).

Cells in the Regular Part $\mathcal{V}^0$

Let

$$\underline{C} \leq h(K) \leq \overline{C} \tag{2.24}$$

for all cells $K \subset \mathcal{V}^0$.

The constants A, $A^\star$, $\underline{C}$ and $\overline{C}$ are the same for all cells K in $\mathcal{V}_{\boldsymbol{a}}$ and $\mathcal{V}^0$.

2.2.3 Geometric Meshes

Consider the mesh in $\mathcal{V}_{\boldsymbol{a}}$. Because of (2.19) and (2.20), the notion of *layers of cells* is well defined. The *terminal layer* $\mathcal{L}_0$ consists of all cells whose closure contains the vertex $\boldsymbol{a}$:

$$K \in \mathcal{L}_0(\boldsymbol{a}) \iff \boldsymbol{a} \in \overline{K}. \tag{2.25}$$

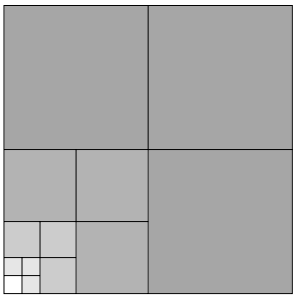


Figure 2.4: Layers in a square. The darkest region is the regular part $\mathcal{V}^0$, the rest of the domain is the neighbourhood $\mathcal{V}_{\boldsymbol{a}}$ of the bottom left corner. The different shades distinguish the different layers. The white cell is the only cell in the terminal layer $\mathcal{L}_0$.

Definition 2.26 (Layer of cells $\mathcal{L}_k$) *The k-th* layer $\mathcal{L}_k$ *is defined as the set of all elements $K \subset \mathcal{V}_{\boldsymbol{a}}$ such that they do not belong to the layers $\mathcal{L}_j$, $0 \leq j \leq k-1$, but touch the layer $\mathcal{L}_{k-1}$*

$$\overline{K} \cap \Big(\bigcup_{K' \in \mathcal{L}_{k-1}} \overline{K'} \Big) \neq \emptyset.$$

Further, analogous to (2.20) *and* (2.22) *hold for all $k \geq 1$:*

$$\frac{\overline{\varkappa}\big(\bigcup_{K \in \mathcal{L}_{k+1}} K\big)}{\overline{\varkappa}\big(\bigcup_{K \in \mathcal{L}_k} \overline{K}\big)} \leq A, \qquad 1 < A^\star \leq \frac{\underline{\varkappa}\big(\bigcup_{K \in \mathcal{L}_{k+1}} K\big)}{\underline{\varkappa}\big(\bigcup_{K \in \mathcal{L}_k} K\big)}.$$

Example 2.27 (Layers) *Figure 2.4 gives an example of four different layers with respect to the bottom left corner $\boldsymbol{a}$ and a regular part in a square.*

Proposition 2.28 (Number of cells in a layer) *The number of cells in a layer is bounded by a constant L_0 for all layers $\mathcal{L}_k$ in $\mathcal{T}$. L_0 depends only on the constants in* (2.16)–(2.21) *and not on the mesh $\mathcal{T} = \{K\}$.*

Proof: It follows from (2.16)–(2.21) that there exists a constant L_0 bounding the number of cells in any layer $\mathcal{L}_k$ [63]. □

The mesh in a neighbourhood $\mathcal{V}_{\boldsymbol{a}}$ is described by the size of the smallest element $K_{\boldsymbol{a}}$ in this part of the mesh. Assume the *geometric grading parameter* $\sigma \leq 1$ and the number of layers $n > 0$. Let $K_{\boldsymbol{a}} \subset \mathcal{V}_{\boldsymbol{a}}$, $\boldsymbol{a} \in \overline{K}_{\boldsymbol{a}}$ and

$$\underline{C}\sigma^n \leq \varkappa(K_{\boldsymbol{a}}) \leq \overline{C}\sigma^n. \tag{2.29}$$

Increasing n results in a growing number of layers (as long as $\sigma < 1$). Using $\sigma = 1$ results in a mesh with the number of layers independent of n.[9] The mesh in the different neighbourhoods $\mathcal{V}_{\boldsymbol{a}}$ can depend on different parameters (σ, n).

The number and size of the cells in the regular part $\mathcal{V}^0$ does not need to fulfil any constraints: the mesh remains the same in this region for different parameters (σ, n).

Definition 2.30 (Geometric mesh) *A mesh $\mathcal{T}$ as described by* (2.16)–(2.29) *(Sections 2.2.2 and 2.2.3) is called a* geometric mesh *in a two dimensional Lipschitz polygon D with n layers and geometric grading parameter σ.*

2.2.4 hp-Approximation Properties

So far, only the cells K of the geometric mesh $\mathcal{T}$ have been discussed. hp-FEM couples mesh refinements with changes in the polynomial degree vector $\boldsymbol{p}$.

Definition 2.31 (hp-FE space in two dimensions) *Let $\mathcal{T} = \{K\}$ be a geometric mesh, geometrically refined towards the corners $\boldsymbol{a} \in \mathcal{C}$ of D, and $m > 0$ a* degree distribution parameter. *Define a* linearly distributed *degree vector $\boldsymbol{p} = \{p_K\}_{K\in\mathcal{T}}$ as follows: For an element (K, p_K) on layer $\mathcal{L}_k$ in a corner neighbourhood $\mathcal{V}_{\boldsymbol{a}}$, define the degree $p_K := \max\{1, \lceil mk \rceil\}$. Additionally, define $p_K = \max_{K\in\mathcal{T}}\{p_K\}$ in the regular part $\mathcal{V}^0$. Then, $\mathcal{S}^{\boldsymbol{p},1}(D, \mathcal{T})$ from* (2.10) *is called an hp-*FE space.

In contrast to Section 2.1.3 (h- and p-FEM), Definition 2.31 prescribes a different polynomial degree p_K for every element $K \in \mathcal{T}$. The hp-extension is typically realised by refining the elements in the terminal layers (h-extension) and by increasing the degree in all elements but those in the terminal layer (p-extension). This results in a degree distribution parameter $m = 1$. If the h-extension in the terminal layer is done via bisection, then the geometric grading factor $\sigma = 1/2$. A more precise algorithm is given in Section 3.2.

[9] This can be used in the neighbourhood of vertices where the solution is known *not* to behave singularly.

Lemma 2.32 *The dimension N of a sequence of hp-FE spaces $\mathcal{S}^{\boldsymbol{p},l}(D,\mathcal{T})$ in two dimensions behaves like $\mathcal{O}(p^3)$ where $p = \max_K \boldsymbol{p}$.*

Proof: According to Proposition 2.28, there are at most L_0 elements per layer $\mathcal{L}_k$.

Typically, an element with degree p has $(p+1)^2$ degrees of freedom associated to it.[10] Therefore, there are $L_0 \cdot (mk+1)^2$ degrees of freedom on one layer. Summing over all layers and using

$$\sum_{i=1}^{n+1} i^2 = \frac{(n+1)(n+2)(2n+3)}{6} \quad \Rightarrow \quad \sum_{k=0}^{n} L_0 \cdot (mk+1)^2 = \mathcal{O}(p^3),$$

where $\lceil mn \rceil = p$. □

The following results holds for hp-FE spaces in two dimensions [8, 31, 97].

Proposition 2.33 (Convergence of hp-FEM in two dimensions) *Let D be a Lipschitz polygon. Let $\{(\mathcal{T}_\sigma^n, \boldsymbol{p}_m^n)\}_n$ be a sequence of hp-mesh-degree combinations, where the meshes $\mathcal{T}_\sigma^n$ are refined towards the corners $\boldsymbol{a} \in \mathcal{C}$ of D with grading parameter $0 < \sigma < 1$ and the degree vectors $\boldsymbol{p}_m^n$ are linearly distributed with slope $m > 0$ (c.f. Definition 2.31). Let u be the solution of the diffusion problem*

$$-\nabla \cdot \underline{A}\nabla u = f \text{ in } D, \qquad u = 0 \text{ on } \Gamma_D, \qquad \boldsymbol{n} \cdot \underline{A}\nabla u = g \text{ on } \Gamma_N$$

with analytic data f, g and $\underline{A}(\boldsymbol{x})$ on $\overline{\Gamma}_N$ and $\overline{D}$ respectively.

Then, for $N = \dim \mathcal{S}^{\boldsymbol{p}_m^n,1}(D,\mathcal{T}_\sigma^n)$,

$$\min_{v \in \mathcal{S}^{\boldsymbol{p}_m^n,1}(D,\mathcal{T}_\sigma^n)} \|u - v\|_{H^1(D)} \le C \exp(-bN^{1/3}),$$

where b depends on m and σ.

Remark 2.34 (Piecewise analytic data) *If f, g and $\underline{A}$ are only piecewise analytic, the previous result still holds, however, the sequence $\{\mathcal{T}_\sigma^n\}_n$ of geometric meshes must then be refined also to the* corners $\boldsymbol{a}$ of the domains of analyticity *of the data. No geometric refinement is necessary, if the domain of analyticity is itself bounded by an analytic curve.*

[10] Using trunk spaces, this can be lessened somewhat. However, the $\mathcal{O}(p^2)$ behaviour remains, *c.f.* Remark 6.7 on page 177.

2.3 Geometric Meshes in Three Dimensions

As in the previous section, the description of the meshes follows [8, 29]. Remark 2.14 also hold in three dimensions. How these meshes are realized algorithmically is shown in Chapter 3.

2.3.1 Domains

The description of the domain D and its associated parts is more involved compared to the situation in two dimensions. Regions close to corners but far from edges and vice-versa have to be distinguished.

Let $D \subset \mathbb{R}^3$ be a bounded Lipschitz polyhedron defined as follows (refer to Example 2.38 and Figure 2.5 below to illustrate the following definitions).

Definition 2.35 (Lipschitz polyhedron in three dimensions) *Let the domain D in $\mathbb{R}^3$ be bounded. At each point $\boldsymbol{a} \in \partial D$, there exists $R_{\boldsymbol{a}} > 0$ and a diffeomorphism $\chi_{\boldsymbol{a}}$ such that the* corner neighbourhood $\mathcal{V}_{\boldsymbol{a}} := D \cap B_{R_{\boldsymbol{a}}}(\boldsymbol{a})$ *is transformed into a neighbourhood of the corner 0 of a cone $\Gamma_{\boldsymbol{a}} = \{\boldsymbol{x} \in \mathbb{R}^3 : \boldsymbol{x}/|\boldsymbol{x}| \in G_{\boldsymbol{a}}\}$ with $G_{\boldsymbol{a}}$ a Lipschitz corner domain of $\mathbb{S}^2$ and $\boldsymbol{a}$ being sent into 0. Assume that the diffeomorphism $\chi_{\boldsymbol{a}}$ is an isometric transformation at $\boldsymbol{a}$.*

The set $\mathcal{C}$ of corners *of D is the set of points $\boldsymbol{a} \in \partial D$ where the corresponding cone $\Gamma_{\boldsymbol{a}}$ is a non-trivial cone (i. e. it is neither a half space nor a wedge).*

An edge e in the set $\mathcal{E}$ of open edges *of D consists of points $\boldsymbol{x} \in e$ such that the local cone $\Gamma_{\boldsymbol{x}}$ is a wedge $\Gamma_e(\boldsymbol{x}) \times \mathbb{R}$ and D is diffeomorphic to this wedge in the neighbourhood $\mathcal{V}_e(\boldsymbol{x}) := D \cap B_{R_{\boldsymbol{x}}}(\boldsymbol{x})$. Here, $\Gamma_e(\boldsymbol{x})$ is a plane sector whose opening is denoted by $\omega_e(\boldsymbol{x})$. Like for two dimensional domains, this opening is intrinsic.*

Definition 2.36 (Edge distance functions r_e and ρ_e) *For $\boldsymbol{x} \in D$, define the distance r_e from the edge e as*

$$r_e(\boldsymbol{x}) := \operatorname{dist}(\overline{e}, \boldsymbol{x}).$$

The distance function ρ_e is defined as follows

1. *if $\overline{e}$ contains no corner, then it is a closed curve and $\rho_e(\boldsymbol{x}) := r_e(\boldsymbol{x})$.*

2. *if $\overline{e}$ contains exactly one corner $\boldsymbol{c} \in \mathcal{C}$, then it is a closed curve with a corner. Define ρ_e such that there holds*

$$r_e(\boldsymbol{x}) = \rho_e(\boldsymbol{x}) r_{\boldsymbol{c}}(\boldsymbol{x}).$$

3. *if $\overline{e}$ contains exactly two corners $\boldsymbol{c}$ and $\boldsymbol{c}' \in \mathcal{C}$, then define ρ_e such that there holds*

$$r_e(\boldsymbol{x}) = \rho_e(\boldsymbol{x}) r_{\boldsymbol{c}}(\boldsymbol{x}) r_{\boldsymbol{c}'}(\boldsymbol{x}).$$

In the neighbourhood $\mathcal{V}_e(\boldsymbol{x})$, the function r_e is equivalent to the radial coordinate in $\Gamma_e(\boldsymbol{x})$. Clearly, ρ_e is equivalent to $r_e/r_{\boldsymbol{c}}$ in $\mathcal{V}_{\boldsymbol{c}}$ and to r_e outside of $\mathcal{V}_{\boldsymbol{c}} \cup \mathcal{V}_{\boldsymbol{c}'}$. The idea of ρ_e is to be able to separate corners and edges with the blow-up of ρ_e close to corners.

Apart from the neighbourhoods $\mathcal{V}_{\boldsymbol{a}}$ and $\mathcal{V}_e(\boldsymbol{x})$ defined above, domains relating corners and edges are necessary.

Definition 2.37 (Corner and edge neighbourhoods)

- Corner-edge neighbourhood $\mathcal{V}_e(\boldsymbol{a})$ including edge: *Let $\boldsymbol{a} \in \mathcal{C}$ be a corner of D. The corners $\boldsymbol{c} = \boldsymbol{c}(e)$ of the spherical domain $G_{\boldsymbol{a}} \subset \mathbb{S}^2$ correspond bijectively to the edges $e \in \mathcal{E}$ such that $\boldsymbol{a} \in \overline{e}$. The function $\rho_e(\boldsymbol{x})$ is equivalent to the radial coordinate at the corner $\boldsymbol{c}(e)$ in $G_{\boldsymbol{a}}$. The domain $\mathcal{V}_e(\boldsymbol{a})$ is defined by the sectorial coordinates*[11] *$(r_{\boldsymbol{a}}, \rho_e, \theta_e)$, where $r_{\boldsymbol{a}} < \varepsilon$, $\rho_e \leq 1$ and θ_e is the angular coordinate in $\mathcal{V}_{\boldsymbol{c}} \subset G_{\boldsymbol{a}}$ is a neighbourhood of the corner $\boldsymbol{c} \in G_{\boldsymbol{a}}$.*
- Corner neighbourhood $\mathcal{V}_{\boldsymbol{a}}^0$ excluding corner-edge neighbourhoods: *Let $\mathcal{V}_{\boldsymbol{a}}^0$ be an open set such that $e \cap \overline{\mathcal{V}_{\boldsymbol{a}}^0} = \emptyset$ for all edges $e \in \mathcal{E}$ and such that*

$$\mathcal{V}_{\boldsymbol{a}} = \mathcal{V}_{\boldsymbol{a}}^0 \cup \Big(\bigcup_{e \in \mathcal{E}_{\boldsymbol{a}}} \mathcal{V}_e(\boldsymbol{a}) \Big).$$

- Edge neighbourhood $\mathcal{V}_e^0$ excluding corner neighbourhoods: *Let $\mathcal{V}_e^0$ be an open set such that no corner or any other edge but e is contained in $\overline{\mathcal{V}_e^0}$. Additionally, $\overline{e}$ is contained in*

$$\overline{\mathcal{V}_e^0} \cup \Big(\bigcup_{\boldsymbol{c} \in \overline{e}} \overline{\mathcal{V}_e(\boldsymbol{c})} \Big)$$

[11] Interpretation as cylindrical coordinates with $\rho_e = r_e/r_{\boldsymbol{c}} \leq 1$ (this is allowed as $r_{\boldsymbol{c}'}$ is nearly constant close to $\boldsymbol{c}$).

Definition 2.37 splits D into the neighbourhoods $\mathcal{V}_e(\boldsymbol{a})$, $\mathcal{V}_{\boldsymbol{a}}^0$, $\mathcal{V}_e^0$ and the *regular part* $\mathcal{V}^0$ (disjoint unions):

$$D = \mathcal{V}^0 \cup \Big(\bigcup_{e\in\mathcal{E}} \mathcal{V}_e^0\Big) \cup \left(\bigcup_{\boldsymbol{a}\in\mathcal{C}} \mathcal{V}_{\boldsymbol{a}}^0 \cup \Big(\bigcup_{e\in\mathcal{E}_{\boldsymbol{a}}} \mathcal{V}_e(\boldsymbol{a})\Big)\right).$$

Example 2.38 (Polyhedral domain) *Let $D = (0,1)^3$ seen from the first octant with $\boldsymbol{a} = (1,1,1)$. Figure 2.5 shows the different neighbourhoods mentioned in Definition 2.37.*

2.3.2 Assumptions on Cells

Consider hexahedral cells K in a mesh $\mathcal{T} = \{K\}$ covering the whole domain D. F_K denotes the bijective map from the *reference cell* $\hat{K} = (-1,1)^3$ onto K: $F_K : \boldsymbol{\xi} \mapsto \boldsymbol{x} = F_K(\boldsymbol{\xi})$.

The cells K in the different regions introduced in Section 2.3.1 are considered separately.

Cells in Edge Neighbourhood $\mathcal{V}_e^0$ excluding Corner Neighbourhoods

Let $K \subset \mathcal{V}_e^0$ and $\overline{K} \cap e = \emptyset$. Denote

$$\underline{\varkappa}(K) = \min_{\boldsymbol{x}\in K} r_e(\boldsymbol{x}), \qquad \overline{\varkappa}(K) = \max_{\boldsymbol{x}\in K} r_e(\boldsymbol{x}). \tag{2.39}$$

The following assumptions on K and F_K hold:

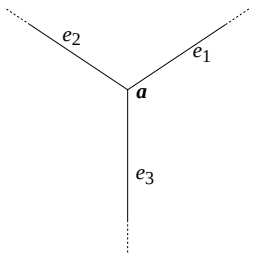
- There exists a constant $A \in (1,\infty)$ such that

$$\frac{\overline{\varkappa}(K)}{\underline{\varkappa}(K)} \le A. \tag{2.40}$$

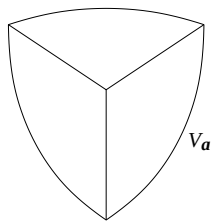
- Assume that the edge e is parallel to the local x_3 axis of the cell K. For any $|\boldsymbol{\alpha}| = m$, $m > 0$ an integer, $h := \overline{\varkappa}(K)$ and any $\boldsymbol{\xi} \in \hat{K}$:

$$\begin{aligned} \left|D^{\boldsymbol{\alpha}}(F_K)_i(\boldsymbol{\xi})\right| &\le C_0 d_0^m (m!) h, \ \text{for } i = 1,2, \\ \left|D^{\boldsymbol{\alpha}}(F_K)_3(\boldsymbol{\xi})\right| &\le C_0 d_0^m (m!) H, \ \text{for } H < H_0, \end{aligned} \tag{2.41}$$

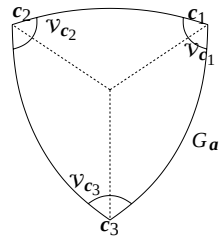
  with C_0, d_0 and H independent of m and h (and H_0 such that $K \subset \mathcal{V}_e^0$). H is a measure for the size of K along e.



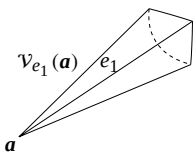
(a) Vertex $\boldsymbol{a}$ and edges e_i, $i = 1, 2, 3$ of $D = (0, 1)^3$.



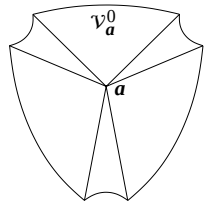
(b) Corner neighbourhood $\mathcal{V}_{\boldsymbol{a}} := D \cap B_{r_{\boldsymbol{a}}}(\boldsymbol{a})$.



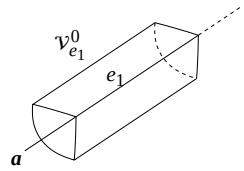
(c) Spherical domain $G_{\boldsymbol{a}} \subset \mathbb{S}^2$ with its corners $\boldsymbol{c}_i$ and the corner neighbourhoods $\mathcal{V}_{\boldsymbol{c}_i} \subset G_{\boldsymbol{a}}$ for $i = 1, 2, 3$.



(d) Corner-edge neighbourhood $\mathcal{V}_{e_1}(\boldsymbol{a})$ including edge.



(e) Corner neighbourhood $\mathcal{V}^0_{\boldsymbol{a}}$ excluding corner-edge neighbourhoods.



(f) Edge neighbourhood $\mathcal{V}^0_{e_1}$ excluding corner neighbourhoods.

Figure 2.5: $D = (0, 1)^3$ seen from the first octant with $\boldsymbol{a} = (1, 1, 1)$. (a) shows the edges meeting in $\boldsymbol{a}$, (b)–(f) depict the domains $\mathcal{V}_{\boldsymbol{a}}$, $G_{\boldsymbol{a}} \subset \mathbb{S}^2$, $\mathcal{V}_{e_1}(\boldsymbol{a})$, $\mathcal{V}^0_{\boldsymbol{a}}$ and $\mathcal{V}^0_{e_1}$.

- The Jacobian determinant is positive and satisfies

$$C_1 H h^2 \leq \det \frac{dF_K}{d\boldsymbol{\xi}} \leq C_2 H h^2, \tag{2.42}$$

  where C_1 and C_2 are independent of h and H.

There exists

$$1 < A^\star \leq \frac{\overline{\varkappa}(K)}{\underline{\varkappa}(K)}. \tag{2.43}$$

The cells K_e touching the edge (*i. e.* $\overline{K}_e \cap e \neq \emptyset$) satisfy (2.41) and (2.42).

Remark 2.44 *The cells in this neighbourhood are needle elements, i. e. they have a very large aspect ratio. In a slice perpendicular to the edge e, the cells look similar as in the vicinity of a corner in a two dimensional domain.*

Cells in Corner-Edge Neighbourhood $\mathcal{V}_e(\boldsymbol{a})$ including Edge

Let $K \subset \mathcal{V}_e(\boldsymbol{a})$ and $\overline{K} \cap e = \emptyset$. Denote

$$\underline{\varkappa}_1(K) = \min_{\boldsymbol{x}\in K} r_{\boldsymbol{a}}(\boldsymbol{x}), \qquad \overline{\varkappa}_1(K) = \max_{\boldsymbol{x}\in K} r_{\boldsymbol{a}}(\boldsymbol{x}), \tag{2.45}$$

$$\underline{\varkappa}_2(K) = \min_{\boldsymbol{x}\in K} \sin\varphi(\boldsymbol{x}), \qquad \overline{\varkappa}_2(K) = \max_{\boldsymbol{x}\in K} \sin\varphi(\boldsymbol{x}), \tag{2.46}$$

where $\sin\varphi(\boldsymbol{x}) := r_e(\boldsymbol{x})/r_{\boldsymbol{a}}(\boldsymbol{x})$. The following assumptions on K and F_K hold:

- There exist constants $A_1, A_2 \in (1, \infty)$ such that

$$\frac{\overline{\varkappa}_1(K)}{\underline{\varkappa}_1(K)} \leq A_1, \tag{2.47}$$

$$\frac{\overline{\varkappa}_2(K)}{\underline{\varkappa}_2(K)} \leq A_2. \tag{2.48}$$

- Assume that the edge e is parallel to the local x_3 axis of the cell K. For any $|\boldsymbol{\alpha}| = m$, $m > 0$ an integer, $h := \overline{\varkappa}_1(K)$, $s := \overline{\varkappa}_2(K)$ and any $\boldsymbol{\xi} \in \hat{K}$:

$$\begin{aligned} \left|D^{\boldsymbol{\alpha}}(F_K)_i(\boldsymbol{\xi})\right| &\leq C_0 d_0^m (m!) h s, \ \text{for } i = 1, 2, \\ \left|D^{\boldsymbol{\alpha}}(F_K)_3(\boldsymbol{\xi})\right| &\leq C_0 d_0^m (m!) h, \end{aligned} \tag{2.49}$$

  with C_0 and d_0 independent of m, h and s.

- The Jacobian determinant is positive and satisfies

$$C_1 h^3 s^2 \leq \det \frac{dF_K}{d\boldsymbol{\xi}} \leq C_2 h^3 s^2, \tag{2.50}$$

  where C_1 and C_2 are independent of h_1 and h_2.

There exist

$$1 < A_1^\star \leq \frac{\overline{\varkappa}_1(K)}{\underline{\varkappa}_1(K)}, \qquad 1 < A_2^\star \leq \frac{\overline{\varkappa}_2(K)}{\underline{\varkappa}_2(K)}. \tag{2.51}$$

The cells K_e touching the edge e but not containing the corner $\boldsymbol{a}$ do not satisfy (2.48) but satisfy the rest of the properties given for $\mathcal{V}_e(\boldsymbol{a})$ above.

Cells in Corner Neighbourhood $\mathcal{V}_{\boldsymbol{a}}^0$ excluding Corner-Edge Neighbourhoods

Let $K \subset \mathcal{V}_{\boldsymbol{a}}^0$ and $\overline{K} \cap \boldsymbol{a} = \emptyset$. Denote

$$\underline{\varkappa}(K) = \min_{\boldsymbol{x} \in K} r_{\boldsymbol{a}}(\boldsymbol{x}), \qquad \overline{\varkappa}(K) = \max_{\boldsymbol{x} \in K} r_{\boldsymbol{a}}(\boldsymbol{x}). \tag{2.52}$$

The following assumptions on K and F_K hold:

- There exists $A \in (1, \infty)$ such that

$$\frac{\overline{\varkappa}(K)}{\underline{\varkappa}(K)} \leq A. \tag{2.53}$$

- For any $|\boldsymbol{\alpha}| = m$, $m > 0$ an integer, $h := \overline{\varkappa}(K)$ and any $\boldsymbol{\xi} \in \hat{K}$:

$$\left| D^{\boldsymbol{\alpha}} (F_K)_i(\boldsymbol{\xi}) \right| \leq C_0 d_0^m (m!) h, \text{ for } i = 1, 2, 3, \tag{2.54}$$

  with C_0 and d_0 independent of m and h.

- The Jacobian determinant is positive and satisfies

$$C_1 h^3 \leq \det \frac{dF_K}{d\boldsymbol{\xi}} \leq C_2 h^3, \tag{2.55}$$

  where C_1 and C_2 are independent of h.

There exists

$$1 < A^\star \leq \frac{\overline{\varkappa}(K)}{\underline{\varkappa}(K)}. \tag{2.56}$$

Remark 2.57 *The cells K in the neighbourhood $\mathcal{V}_{\boldsymbol{a}}^0$ do not have a large aspect ratio and their size depends on the distance $r_{\boldsymbol{a}}$ from the corner $\boldsymbol{a}$.*

Cells in the Regular Part $\mathcal{V}^0$

Let $K \subset \mathcal{V}^0$. Denote

$$\varkappa(K) = \min_{e \in \mathcal{E}} r_e(\boldsymbol{x}). \tag{2.58}$$

The following assumptions on K and F_K hold:

- For any $|\boldsymbol{\alpha}| = m$, $m > 0$ an integer, $h := \varkappa(K)$ and any $\boldsymbol{\xi} \in \hat{K}$:

$$\left| D^{\boldsymbol{\alpha}} (F_K)_i(\boldsymbol{\xi}) \right| \leq C_0 d_0^m (m!) h, \text{ for } i = 1, 2, 3, \tag{2.59}$$

  with C_0 and d_0 independent of m and h.

- The Jacobian determinant is positive and satisfies

$$C_1 h^3 \leq \det \frac{dF_K}{d\boldsymbol{\xi}} \leq C_2 h^3, \tag{2.60}$$

  where C_1 and C_2 are independent of h.

Vertex Cells

Let $\boldsymbol{a} \in \overline{K}_{\boldsymbol{a}}$ where $\boldsymbol{a} \in \mathcal{C}$ is a corner of the domain D. Denote

$$\varkappa(K_{\boldsymbol{a}}) = \operatorname{diam} K_{\boldsymbol{a}}. \tag{2.61}$$

The assumptions (2.48)–(2.50), (2.53) and (2.54) on K and F_K hold.

The constants A, A_1, A_2, $A^\star$, $A_1^\star$, $A_2^\star$, C_0, d_0, C_1 and C_2 are the same for all cells K in the mesh $\mathcal{T}$.

2.3.3 Geometric Meshes

Similarly to the two dimensional case, the assumptions in Section 2.3.2 allow the definition of *layers of cells* in the mesh $\mathcal{T}$. The *terminal layer* $\mathcal{L}_0$ in the neighbourhoods $\mathcal{V}_e(\boldsymbol{a})$, $\mathcal{V}_{\boldsymbol{a}}^0$ and $\mathcal{V}_e^0$ consists of all cells whose closure contain the 'main feature' of the neighbourhood: the edge e and the corner $\boldsymbol{a}$ respectively.

The layers $\mathcal{L}_k$, $0 \leq k \leq n$ are defined similarly to Definition 2.26. Assume a *geometric grading parameter* $\sigma \leq 1$.

Layers in Edge Neighbourhood $\mathcal{V}_e^0$ excluding Corner Neighbourhoods

The mesh is geometric (in the two dimensional sense given in Definition 2.30) perpendicular to the edge e. The cells K_e in the terminal layer $\mathcal{L}_0$ touching the edge e are constrained by

$$\overline{\varkappa}(K_e) \leq \overline{C}\sigma^n. \tag{2.62}$$

In the direction along the edge e, several *levels* are allowed. Nonetheless, the number of cells in a layer is bounded and the total number of cells in this neighbourhood is of order $\mathcal{O}(n)$.

Layers in Corner-Edge Neighbourhood $\mathcal{V}_e(\boldsymbol{a})$ including Edge

The mesh in this neighbourhood is geometric perpendicular to the edge (arranged in layers) and along the edge (arranged in levels). The cells K fulfil

$$\overline{\varkappa}_2(K_e) \leq \overline{C}\sigma^n, \qquad \underline{C}\sigma^n \leq \min_{K \subset \mathcal{V}_e(\boldsymbol{a})} \overline{\varkappa}_1(K) \leq \overline{C}\sigma^n. \tag{2.63}$$

The total number of cells in this neighbourhood is of order $\mathcal{O}(n^2)$.

Layers in Corner Neighbourhood $\mathcal{V}_{\boldsymbol{a}}^0$ excluding Corner-Edge Neighbourhoods

The mesh is geometric in the distance $r_{\boldsymbol{a}}$ to the corner $\boldsymbol{a}$ with

$$\underline{C}\sigma^n \leq \min_{K \subset \mathcal{V}_{\boldsymbol{a}}^0} \overline{\varkappa}(K) \leq \overline{C}\sigma^n. \tag{2.64}$$

The total number of cells in this neighbourhood is of order $\mathcal{O}(n)$.

Regular Part $\mathcal{V}^0$

In the regular part, there are no layers and the number of cells is independent of n.

Vertex Cells

The size of the cells $K_{\boldsymbol{a}}$ fulfils

$$\underline{C}\sigma^n \leq \operatorname{diam}(K_{\boldsymbol{a}}) \leq \overline{C}\sigma^n. \tag{2.65}$$

Definition 2.66 (Geometric mesh) *A mesh $\mathcal{T}$ as described by* (2.39)–(2.65) *(Sections 2.3.2 and 2.3.3) is called a* geometric mesh *in a three dimensional Lipschitz polyhedron D with n layers and geometric grading parameter σ.*

2.3.4 hp-Approximation Properties

So far, only the cells K of the mesh $\mathcal{T}$ have been discussed. Now, we address the combination of mesh and polynomial degree.

Definition 2.67 (hp-FE space in three dimensions) *On a geometric mesh $\mathcal{T} = \{K\}$, a* linearly distributed *degree vector $\boldsymbol{p}$ with slope $m > 0$ is defined as follows:*

- *The elemental degree $\boldsymbol{p}_K \in \mathbb{N}_0^3$ is allowed to be anisotropic in all elements $(K, \boldsymbol{p}_K)$: $\boldsymbol{p}_K = (p_{1,K}, p_{2,K}, p_{3,K})$.*
- Vertex elements $(K_{\boldsymbol{a}}, p_{K_{\boldsymbol{a}}})$: *define* $\boldsymbol{p}_K = (1, 1, 1)$.
- Regular part $\mathcal{V}^0$: *define* $\boldsymbol{p}_K = \max_{K\in\mathcal{T}}\{\boldsymbol{p}_K\} =: (p_{\max}, p_{\max}, p_{\max})$.
- Corner neighbourhood $\mathcal{V}_{\boldsymbol{a}}^0$ excluding corner-edge neighbourhoods: *define*

  $$\boldsymbol{p}_K = (\max\{1, \lceil mk\rceil\}, \max\{1, \lceil mk\rceil\}, \max\{1, \lceil mk\rceil\})$$

  for an element $(K, \boldsymbol{p}_K)$ on layer $\mathcal{L}_k$.
- Edge neighbourhood $\mathcal{V}_e^0$ excluding corner neighbourhoods: *without loss of generality, let the local x_3 axis be parallel to the edge e. An element on layer $\mathcal{L}_k$ has* $\boldsymbol{p}_K = (\max\{1, \lceil mk\rceil\}, \max\{1, \lceil mk\rceil\}, p_{\max})$, $\boldsymbol{p}_K \geq 1$.

- Corner-edge neighbourhood $\mathcal{V}_e(\boldsymbol{a})$ including edge: *again, let the local x_3 axis be parallel to the edge e. An element in layer $\mathcal{L}_k$ and level $\mathcal{L}_l$ has* $\boldsymbol{p}_K = (\max\{1, \lceil mk \rceil\}, \max\{1, \lceil mk \rceil\}, \max\{1, \lceil ml \rceil\})$.

Then, $\mathcal{S}^{\boldsymbol{p},1}(D, \mathcal{T})$ from (2.10) *is called an hp-*FE space.

Lemma 2.68 *The dimension N of a sequence of hp-FE spaces $\mathcal{S}^{\boldsymbol{p},l}(D, \mathcal{T})$ in three dimensions behaves like $\mathcal{O}(p^s)$ where $p = \max\{p_{\max,i}\}_{i=1}^3$ and $\boldsymbol{p}_{\max} = \max_K \boldsymbol{p}_K$. $s = 4$ if the geometric mesh is only refined towards vertex singularities and $s = 5$ if there are also edge singularities taken into account for the mesh.*

Proof: Analogous to the proof of Lemma 2.32 by taking into account the layers and levels along singular edges for $s = 5$. □

The hp-extension is typically realised by refining (anisotropically) the elements in the terminal layers (h-extension) and by increasing (anisotropically) the degree in all elements but those in the terminal layer (p-extension). A more precise algorithm is given in Section 3.2.

The following result holds for hp-FE spaces in three dimensions [8, 32, 97].

Proposition 2.69 (Convergence of hp-FEM in three dimensions) *Let D be a Lipschitz polyhedron. Let $\{(\mathcal{T}_\sigma^n, \boldsymbol{p}_m^n)\}_n$ be a sequence of hp-mesh-degree combinations, where the meshes $\mathcal{T}_\sigma^n$ are refined towards the corners $\boldsymbol{a} \in \mathcal{C}$ and the edges $e \in \mathcal{E}$ of D with grading parameter $0 < \sigma < 1$ and the degree vectors $\boldsymbol{p}_m^n$ are linearly distributed with slope $m > 0$ (c. f. Definition 2.67). Let u be the solution of the diffusion problem*

$$-\nabla \cdot \underline{A} \nabla u = f \text{ in } D, \qquad u = 0 \text{ on } \Gamma_D, \qquad \boldsymbol{n} \cdot \underline{A} \nabla u = g \text{ on } \Gamma_N$$

with analytic data f, g and $\underline{A}(\boldsymbol{x})$ on $\overline{\Gamma}_N$ and $\overline{D}$ respectively.

Then, for $N = \dim \mathcal{S}^{\boldsymbol{p}_m^n,1}(D, \mathcal{T}_\sigma^n)$,

$$\min_{v \in \mathcal{S}^{\boldsymbol{p}_m^n,1}(D,\mathcal{T}_\sigma^n)} \|u - v\|_{H^1(D)} \leq C \exp(-bN^{1/5}),$$

where b depends on m and σ.

Remark 2.70

- *It is not necessary to distribute the degrees $\boldsymbol{p}_K$ anisotropically as described in Definition 2.67—isotropic degrees p_K or even a uniform degree p would suffice (even for the exponential convergence of Proposition 2.69). However, given a desired accuracy, using a non-uniform, anisotropic $\boldsymbol{p}_K$ saves a considerable amount of degrees of freedom.*

- *Note that only piecewise analyticity of the data f, g and $\underline{A}$ is required if $\mathcal{T}_\sigma^n$ is refined accordingly, c. f. Remark 2.34.*

2.4 Further Examples (Formulations and Operators)

The following two subsections give some outlook to other Finite Element Methods which (at least partially) fit into this setting. Importantly, the methods discussed fit into to software design which is presented in Part III.

2.4.1 Discontinuous Galerkin Finite Element Methods

Reaction Diffusion

A typical variational form for discontinuous Galerkin FEM (DGFEM) of the problem described in Section 1.3.1 reads [112]:
Find $u_N \in V_N$ such that

$$B_h^{\pm}(u_N, v_N) = l_h^{\mp}(v_N) \quad \forall v_N \in V_N, \tag{2.71}$$

where

$$\begin{aligned} B_h^{\pm}(u, v) := & \sum_{K\in\mathcal{T}} \int_K (\nabla v \cdot \underline{A}\nabla u + cuv)\, d\boldsymbol{x} \\ & - \sum_{e\in\mathcal{E}_{\text{int},D}} \int_e \left(\langle \underline{A}\nabla u\rangle \cdot [v] \mp [u] \cdot \langle \underline{A}\nabla v\rangle\right) d\boldsymbol{s} \\ & + \sum_{e\in\mathcal{E}_{\text{int},D}} \int_e \delta\,[u][v]\, d\boldsymbol{s}, \\ l_h^{\mp}(v) := & \sum_{K\in\mathcal{T}} \int_K f\, v\, d\boldsymbol{x} \\ & + \int_{\Gamma_N} g_N v\, d\boldsymbol{s} \mp \int_{\Gamma_D} (\underline{A}\nabla v \cdot \boldsymbol{n}_D) g_D\, d\boldsymbol{s} + \int_{\Gamma_D} \delta g_D v\, d\boldsymbol{s} \end{aligned} \tag{2.72}$$

and $\delta = \delta(p, h)$ the so-called discontinuity stabilisation function. Here, the non-homogeneous Dirichlet boundary condition g_D on Γ_D is weakly enforced, *i. e.*, it

is not part of the DGFE space. The finite dimensional space V_N is defined as

$$V_N := \{u \in L^2(D) : u|_K \circ F_K \in \mathcal{P}_{p_K}(\hat{K}), K \in \mathcal{T}\} = \mathcal{S}^{\boldsymbol{p},0}(D, \mathcal{T}). \qquad (2.73)$$

In contrast to the FE spaces defined in Definitions 2.31 and 2.67, (2.73) defines discontinuous functions.

The DGFEM defined by the bilinear and linear forms in (2.72) are symmetric ('$-$' version) and non-symmetric ('$+$' version). For sufficiently smooth data (f, g_N and g_D), the functionals in (2.72) are well-defined and (2.71) is consistent. The bilinear forms $B_h^{\pm}(u, v)$ are continuous and coercive over appropriately defined energy spaces and $l_h^{\mp}(v)$ is continuous. Therefore, this method fits into the framework of Section 1.1. The approximation results for the DGFEM are given in [112].

Stokes

Let $D \subset \mathbb{R}^3$ be a bounded domain. The Stokes problem reads:

$$-\nu\Delta\boldsymbol{u} + \nabla p = \boldsymbol{f}, \qquad \operatorname{div}\boldsymbol{u} = 0 \quad \text{in } D \qquad (2.74)$$

with the boundary conditions

$$\boldsymbol{u} = \boldsymbol{g} \quad \text{on } \partial D.$$

Here, $\boldsymbol{u}$ denotes the velocity field, ν the viscosity and p the pressure. The inf-sup condition [20, 61]

$$\inf_{0\neq L^2(D)/\mathbb{R}} \sup_{0\neq \boldsymbol{v}\in H_0^1(D)^3} \frac{-\int_D q \operatorname{div}\boldsymbol{v}\, d\boldsymbol{x}}{|\boldsymbol{v}|_1 \, \|q\|_0} \geq C_D > 0$$

guarantees a unique solution $(\boldsymbol{u}, p) \in H_0^1(D)^3 \times L^2(D)/\mathbb{R}$ of (2.74). Here, $|.|_1$ denotes the H^1-semi-norm.

The Stokes problem is a saddle-point problem. Classically, mixed FEM are used to approximately solve (2.74): degree p for the velocities and $p-k$ for the pressure. Conforming methods need $k = 2$ and discontinuous methods $k = 1$ [108]. [96] shows the stability of pressure stabilised hp-DGFEM for $k = 0$ (so-called equal order methods).

The framework for vector valued problems given in Section 7.1 allows mixed methods ($k \neq 0$) as long as the underlying mesh $\mathcal{T}$ is the same in all components of the vector valued FE spaces.

2.4.2 Non-local Operators: Boundary Element Methods

Boundary Element Methods (BEM) are used to discretise boundary integral equations which can be derived from a homogeneous partial differential equation $\mathcal{L}u = 0$ with Neumann (u_N) or Dirichlet (u_D) boundary conditions [93]. A boundary integral equation in general form reads:
Find $u \in H^{s/2}(\Gamma)$ [1] such that

$$(\mathcal{A}(u), v) =: a(u, v) = (f, v) \quad \forall v \in H^{s/2}(\Gamma), \tag{2.75}$$

where $f \in H^{-s/2}(\Gamma)$, $s \in \{0, \pm 1\}$ and $\mathcal{A}(.)$ is one of the following integral operators:

$$u_D(\boldsymbol{x}) = \mathcal{V}(\sigma)(\boldsymbol{x}) := \int_\Gamma \sigma(\boldsymbol{y}) k(\boldsymbol{x}, \boldsymbol{y})\, ds_{\boldsymbol{y}},$$

$$u_D(\boldsymbol{x}) = -1/2\varphi(\boldsymbol{x}) + \mathcal{K}(\varphi)(\boldsymbol{x}) := -1/2\varphi(\boldsymbol{x}) + \int_\Gamma \varphi(\boldsymbol{y}) \gamma_{1,\boldsymbol{y}} k(\boldsymbol{x}, \boldsymbol{y})\, ds_{\boldsymbol{y}},$$

$$u_N(\boldsymbol{x}) = 1/2\sigma(\boldsymbol{x}) + \mathcal{K}'(\sigma)(\boldsymbol{x}) := 1/2\sigma(\boldsymbol{x}) + \mathrm{pv} \int_\Gamma \sigma(\boldsymbol{y}) \gamma_{1,\boldsymbol{x}} k(\boldsymbol{x}, \boldsymbol{y})\, ds_{\boldsymbol{y}},$$

$$u_N(\boldsymbol{x}) = -\mathcal{W}(\varphi)(\boldsymbol{x}) := -\mathrm{fp} \int_\Gamma \varphi(\boldsymbol{y}) \gamma_{1,\boldsymbol{x}} \gamma_{1,\boldsymbol{y}} k(\boldsymbol{x}, \boldsymbol{y})\, ds_{\boldsymbol{y}}.$$

$\gamma_{1,\boldsymbol{x}}$ denotes the normal derivative with respect to $\boldsymbol{x}$ and $\sigma(\boldsymbol{x})$ and $\varphi(\boldsymbol{x})$ are used to describe the solution $u(\boldsymbol{x})$ using the kernel $k(\boldsymbol{x}, \boldsymbol{y})$ of the PDE $\mathcal{L}u = 0$:

$$u(\boldsymbol{x}) = \int_\Gamma \sigma(\boldsymbol{y}) k(\boldsymbol{x}, \boldsymbol{y})\, ds_{\boldsymbol{y}} \quad \text{or} \quad u(\boldsymbol{x}) = \int_\Gamma \varphi(\boldsymbol{y}) \gamma_{1,\boldsymbol{y}} k(\boldsymbol{x}, \boldsymbol{y})\, ds_{\boldsymbol{y}}.$$

The kernel $k(\boldsymbol{x}, \boldsymbol{y})$ fulfils $\mathcal{L}k(., \boldsymbol{y}) = \delta(\boldsymbol{y})$. (f, v) denotes the L^2 duality pairing extended to $H^{-s/2}(\Gamma) \times H^{s/2}(\Gamma)$ by continuity. $a(u, v)$ is the bilinear form $\int_\Gamma \mathcal{A}(u)(\boldsymbol{x}) v(\boldsymbol{x})\, ds$ fulfilling

- continuity,
- Gårding inequality: There exists $c > 0$ and a compact operator

  $$K : H^{s/2}(\Gamma) \to H^{-s/2}(\Gamma)$$

  such that $\forall u \in H^{s/2}(\Gamma)$:

  $$a(u, u) + (Ku, u) \geq c \|u\|^2_{H^{s/2}(\Gamma)}.$$

- injectivity in $H^{s/2}(\Gamma)$: $a(u,u) = 0 \Rightarrow u = 0$.

Then, (2.75) admits a unique solution $\forall f \in H^{-s/2}(\Gamma)$ [93]. This method fits into the framework of Section 1.1 and also into the software desing presented in Part III.

3 Algorithmic Realization of hp-Finite Element Spaces in $\mathbb{R}^3$

In two dimensions, to resolve corner singularities of elasticity and Maxwell problems at an *exponential rate of convergence* in polygons, *geometric vertex meshes* are sufficient. To resolve boundary layers due to singular perturbations as, *e. g.*, in viscous, incompressible flows and reaction diffusion problems, *geometric boundary layer meshes* are necessary.

In three dimensions, *geometric vertex and edge meshes* are necessary to resolve the singularities which arise in elasticity and Maxwell problems at an exponential rate of convergence. To capture the features of viscous, incompressible flows and reaction diffusion problems, again *geometric boundary layer meshes* are needed. Solutions of eddy current interface problems also exhibit these boundary layers at internal interfaces. A class of geometric edge, vertex and face meshes in general polyhedra in $\mathbb{R}^3$ are introduced in Section 3.1.

This whole chapter is devoted to the algorithmic realisation of mesh-degree combinations $(\mathcal{T}, \boldsymbol{p})$ used in hp-Finite Element Methods to get exponential convergence. The implementation, which closely follows these guidelines, is described in Part III. The main results presented in this chapter are:

1. an *algorithm to create geometric vertex, edge and boundary layer meshes* built from trilinearly mapped hexahedra in arbitrary combination in any polyhedron in $\mathbb{R}^3$ with a proper polynomial degree distribution as introduced in Chapter 2.

2. a *generic assembly procedure* for irregular meshes with arbitrary, anisotropic polynomial degree distribution.

The first result is presented in the first two sections and the second one in the third section. The last section gives some numerical examples with convergence plots and run-time measurements.

Contrary to popular believe, we are convinced that it is beneficial to work *only with hexahedral meshes:*

- Shape functions, interpolants and S matrices (*c. f.* Section 3.3.5) have *tensor product structure*. Also, it is obvious what the effect of anisotropically reducing the local polynomial degree in one direction is—which is not the case in a tetrahedron.
- It is straight forward to mesh special engineering features like thin plates, thin films or coatings. Meshing such thin structures with a good tetrahedral mesh is almost impossible.
- Geometric refinements towards a geometric feature like an edge or face are simple to achieve.

The *drawback of hexahedral meshes* is that local mesh refinements introduce irregularities in the mesh. It is not straight forward to handle these in an anisotropically refined mesh with general, anisotropic local polynomial degrees. However, we have solved these problems.

The main task treated in the section devoted to the *assembly procedure* (Section 3.3) is to form *globally continuous basis functions* from local shape functions (defined on the cells of the mesh). Problems arise across cell boundaries where the mesh is irregular. In this case, the shape functions in the smaller cell have to be constrained by their larger neighbours. There are different methods to enforce these constraints [14, 15, 41, 42, 43, 47, 50, 68]—the one shown here is flexible (allows for anisotropic refinement in arbitrary number and order and an anisotropic, arbitrary degree) and still remains fast (pre-computed coefficients used during the assembling).

3.1 Creating Geometric Meshes in Three Dimensions

In the presence of singularities or boundary layers, geometric meshes (and degrees) adapted to the specific situation have to be used in order to get exponential con-

vergence rates. The geometric grading factor $\sigma \in (0, 1)$ and the number of layers n characterise the meshes. Refer to [8, 83, 85, 98, 109] and the references therein on how to chose these parameters and Chapter 2 for a theoretical description of the meshes. We only work with $\sigma = 1/2$ for implementational reasons.

After a motivation of geometric meshes, a rough description of geometric vertex, edge and boundary layer meshes is given. The geometric vertex and edge meshes fit into the setting of Section 2.3, the geometric boundary layer meshes are not considered there. Finally, we present an algorithm to create these meshes.

3.1.1 Exponential Convergence Needs Geometric Meshes

The use of geometric meshes in Chapter 2 is motivated below such that a simple refinement rule can be drawn as conclusion. This rule is the basis for the algorithms below.

Let u be a function we try to approximate with hp-FEM. The bounded domain of interest is $D \subset \mathbb{R}^d$. u is analytic in $\overline{D} \setminus S$. Here, S is the set of 'singularities' (corners, edges and faces—typically on the boundary of D but possibly also in the interior of the domain in case of jumping coefficients). In a cell $K \in \mathcal{T}$ away from S, $u|_K$ is analytic. Therefore, it can be approximated exponentially with polynomials of increasing order p.

Example 3.1 (One dimension) *Let $D = (0, 1)$ and $u = r^\lambda$ (a model for the singularities arising in one dimensional problems). u is analytic in $\overline{D} \setminus S = (0, 1]$. In the cell $K = (1/2, 1)$, u can be approximated exponentially by polynomials of order p:*

$$\inf_{v \in \mathcal{P}_p} \|u - v\|_{L^\infty(K)} \leq Ce^{-bp}.$$

Other norms than $L^\infty(K)$, e. g. $L^2(K)$, are possible too.

Generally, the convergence rate b depends on

$$\frac{\operatorname{diam} K}{\operatorname{dist}(K, S)}. \tag{3.2}$$

The *basic mesh design principle* for exponential convergence is to keep the ratio (3.2) constant over successive mesh refinements. Consequently, the convergence rate b is also constant. The *cells in the terminal layer* touching S do not need any special treatment as their size is exponentially decreasing with successive refinements (and so does their error contribution).

In three dimensions, the basic idea stays untouched, only the interpretation of $\operatorname{diam} K$ in (3.2) has to be changed. The set of singularities S is anisotropic (*i. e.* contains corners and edges). Therefore, the geometric mesh must also be anisotropic and aligned to S. This keeps

$$\frac{\operatorname{diam}^{\perp} K}{\operatorname{dist}(K, S)}$$

constant. Here,

$$\operatorname{diam}^{\perp} K = \begin{cases} \operatorname{diam} K & \text{if } K \subset \mathcal{V}_{\boldsymbol{a}}^0, \\ \text{diameter of largest inscribed sphere} & \text{if } K \subset \mathcal{V}_e(\boldsymbol{a}) \cup \mathcal{V}_e^0. \end{cases}$$

Similarly for two or three dimensional problems with a boundary layer.

Conclusion: A geometric mesh is refined by subdividing the elements in the terminal layer aligned to the singularities in S. Simultaneously, the polynomial degree is increased in all other elements. In addition, if an element is subdivided anisotropically, the polynomial degree is increased in those directions which are not broken. This ensures exponential convergence in the presence of singularities in three dimensions.

3.1.2 Description of Geometric Vertex, Edge and Boundary Layer Meshes

A mesh $\mathcal{T}$ in a domain D can consist of various regions where different refinement strategies have to be used. Three basic strategies can be distinguished:

- A *geometric boundary layer mesh* $\mathcal{T}$ (*c. f.* Figure 3.1 on the left) is given by a one dimensional geometric mesh $\mathcal{T}^1$ with grading factor σ and n layers and a two dimensional domain Q:

  $$\mathcal{T} = \{I \times Q : I \in \mathcal{T}^1\}.$$

  The meshes in the middle and on the right of Figure 3.1 are geometric boundary layer meshes towards two and three faces respectively.

- A *geometric edge mesh* $\mathcal{T}$ (*c. f.* Figure 3.2 on the left) is given by a two dimensional geometric mesh $\mathcal{T}^2$ (an irregular vertex mesh refined towards

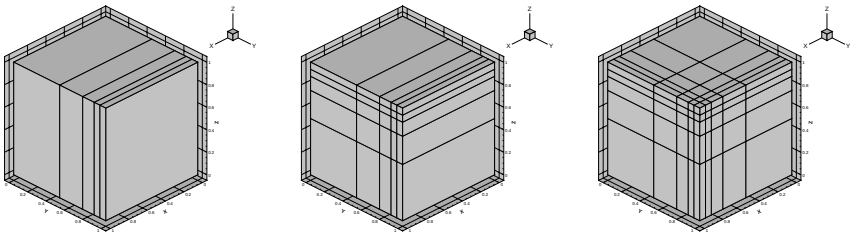


Figure 3.1: Geometric boundary layer mesh with grading factor $\sigma = 1/2$ and 5 layers towards one (left), two (middle) and three faces (right).

one corner with grading factor σ and n layers) and a one dimensional interval I:

$$\mathcal{T} = \{Q \times I : Q \in \mathcal{T}^2\}.$$

The meshes in the middle and on the right of Figure 3.2 are geometric edge meshes towards two and three edges respectively.

- A *geometric vertex mesh* (*c. f.* Figure 3.3) is an irregular vertex mesh refined towards one corner with grading factor σ and n layers.

The right most meshes in Figures 3.1 and 3.2 (geometric boundary layer mesh and geometric edge mesh respectively) look the same from 'outside'. However, looking inside the domain by dropping $1/8$ of the cube reveals the differences, *c. f.* Figure 3.4.

3.1.3 Algorithmic Realization

Algorithm 3.1 is a simple algorithm which can be used to geometrically refine meshes in any bounded polyhedral domain given the following information and tools:

- a hexahedral initial partition $\mathcal{T}^1$, (3.3)
- *a-priori information* which vertex, edge (or even face) is singular (*i. e.* a singularity of the solution does or might occur there),

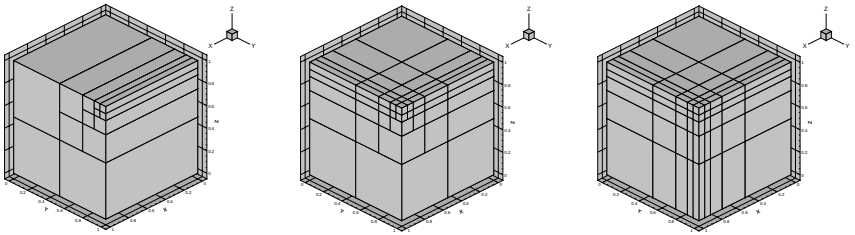
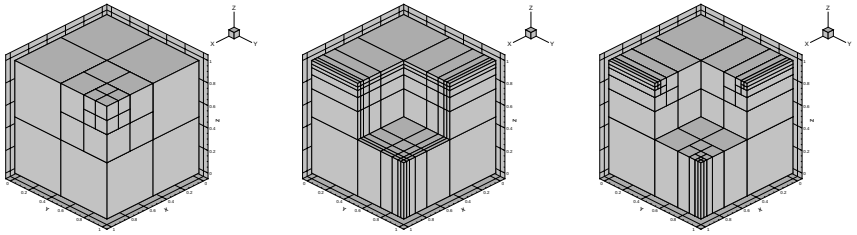


Figure 3.2: Geometric edge mesh with grading factor $\sigma = 1/2$ and 5 layers towards one (left), two (middle) and three edges (right).



(a) Geometric boundary layer mesh towards three faces.

(b) Geometric edge mesh towards three edges.

Figure 3.3: Geometric vertex mesh with grading factor $\sigma = 1/2$ and 4 layers.

Figure 3.4: Meshes with grading factor $\sigma = 1/2$ and 6 layers. Here, $1/8$ of the cube is cut away to reveal details inside the domain.

- means to subdivide every given hexahedron in one of the seven possibilities shown in Figure 3.15.

Algorithm 3.1 computes a cell-wise refinement indicator $\delta\boldsymbol{l} = (\delta l_x, \delta l_y, \delta l_z)$ denoting what changes should be made to the current cell K of the mesh $\mathcal{T}$. Values of 0 in any component of $\delta\boldsymbol{l}$ mean 'no change'. A value of 1 in δl_x means that the x axis of the hexahedron should be broken, similarly $\delta l_y = 1 \Rightarrow$ break y axis and $\delta l_z = 1 \Rightarrow$ break z axis. The eight possibilities offered by $\delta l_i \in \{0, 1\}$ are covered

Loop over all cells K in the mesh $\mathcal{T}$:

- Let $\delta \boldsymbol{l} = (0, 0, 0)$ (the subdivision indicators for the three directions).
- Loop over all vertices of the cell:
  - If a vertex is marked as singular, set $\delta \boldsymbol{l} = (1, 1, 1)$.
- Loop over all edges of the cell:
  - If an edge is marked as singular, set the corresponding two entries in $\delta \boldsymbol{l}$ to 1:
    if the edge is parallel to the x axis, set $\delta l_y = \delta l_z = 1$,
    if the edge is parallel to the y axis, set $\delta l_x = \delta l_z = 1$,
    if the edge is parallel to the z axis, set $\delta l_x = \delta l_y = 1$.
- Loop over all faces of the cell:
  - If a face is marked as singular, set the corresponding entry in $\delta \boldsymbol{l}$ to 1: if the face is perpendicular to the i axis, set $\delta l_i = 1, \forall i \in \{x, y, z\}$.
- Refine the cell K with the refinement indicator $\delta \boldsymbol{l}$.

Algorithm 3.1: Geometric, anisotropic mesh refinement in three dimensions. Given a hexahedral mesh $\mathcal{T}$ of a bounded polyhedron D and a marking of singular vertices, edges and faces, this algorithm applies one anisotropic refinement step and returns the resulting mesh $\mathcal{T}'$.

by the seven subdivisions in Figure 3.15 and 'no subdivision'. Note that Algorithm 3.1 does not change the polynomial degrees. This is done in Algorithm 3.2 below.

Remark 3.4 (Algorithm 3.1)

- *The algorithm is also applicable in two dimensions with obvious changes.*
- *If a vertex in cell K is marked as singular, then the loops over the edges and the faces can be economised because all entries of $\delta \boldsymbol{l}$ are already set to 1.*
- *The notion of 'x axis' etc. is local to the cell.*
- *The algorithm also works for geometric meshes towards internal vertices, edges or faces. This is needed in eddy current interface problems where boundary layers also arise at internal boundaries.*
- *The requirement* (3.3) *is not essential as shown by the result below.*

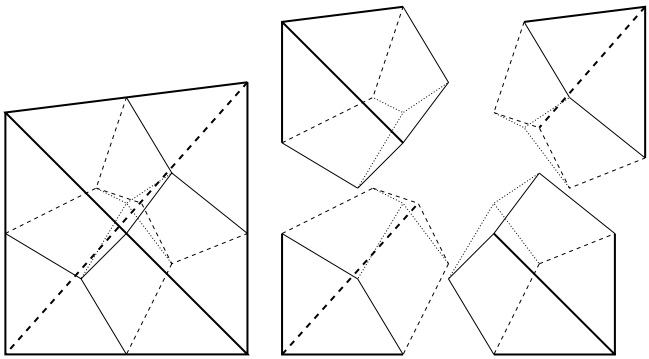


Figure 3.5: Breaking a tetrahedron into four hexahedra. The thick lines show the initial tetrahedron. The dotted lines represent new edges in the interior of the tetrahedron while all other new edges are on the surface of the tetrahedron. The pictures on the right show the four individual hexahedra.

- *The generated meshes are 1-irregular (only singly constrained nodes).*

Proposition 3.5 *Given any bounded polyhedral domain $D \subset \mathbb{R}^3$, there exists a shape-regular, hexahedral initial mesh $\mathcal{T}^1$ in D.*

Proof: It is possible to mesh any bounded polyhedral domain $D \subset \mathbb{R}^3$ with shape-regular tetrahedra.[1] Any tetrahedron can be broken into four trilinearly mapped hexahedra as shown in Figure 3.5. Every face of the tetrahedron is broken into four quadrilaterals by introducing edges joining the edge-midpoints and the centre of gravity of the face. An additional vertex is introduced in the middle of the tetrahedron and linked with the face-centres.

Breaking each tetrahedron into four hexahedra creates a shape-regular, hexahedral mesh. The shape-regularity is preserved by choosing edge mid-points and centres of gravity. □

Example 3.6 *In simple geometries, it is beneficial to create the meshes by hand: A tetrahedral mesh in the cube $(-1,1)^3$ needs 6 tetrahedra resulting in a mesh*

[1] There even exist good algorithms and software for this task to create high quality tetrahedral meshes.

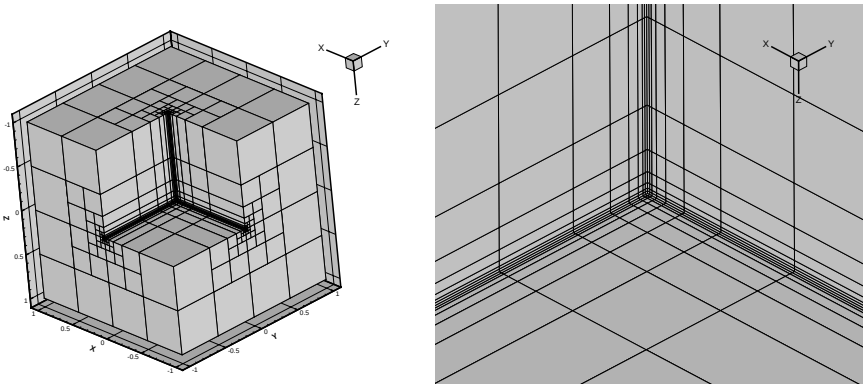


Figure 3.6: Edge mesh in the Fichera corner. The right picture shows an enlarged version centred at the origin.

with 24 hexahedra using the procedure described in Proposition 3.5. However, a hand-made mesh needs only one hexahedron. All meshes in the present thesis are hand-made. In addition, in the hand-made mesh, the reference hexahedron $(0,1)^3$ *is affinely mapped to the physical hexahedron whereas in the automatically generated mesh, all hexahedra are trilinearly mapped. In thin geometries (like a plate), a hand-made hexahedral mesh is even more superior.*

Example 3.7 *Algorithm 3.1 used on* $D = (-1,1)^3 \setminus (-1,0)^3$ *(the so called 'Fichera corner') creates a mesh as shown in Figure 3.6. The three reentrant edges and the reentrant corner (at the origin) were marked as singular.*

3.2 Creating hp-Finite Element Spaces

Algorithm 3.2 shows a simple extension of Algorithm 3.1 to hp-refine mesh-degree combinations $(\mathcal{T}, \boldsymbol{p})$. In addition to the refinement indicator $\delta\boldsymbol{l}$ computed by Algorithm 3.1, Algorithm 3.2 computes a degree indicator $\delta\boldsymbol{p} = (\delta p_x, \delta p_y, \delta p_z)$. Similarly to $\delta\boldsymbol{l}$, $\delta\boldsymbol{p}$ indicates what changes should be made to the polynomial degree $\boldsymbol{p}_K$ of the element $(K, \boldsymbol{p}_K)$ of the current mesh-degree combination $(\mathcal{T}, \boldsymbol{p})$.

Loop over all elements $(K,\ \boldsymbol{p}_K)$ in the mesh-degree combination $(\mathcal{T},\ \boldsymbol{p})$:

- Determine the subdivision indicator $\delta\boldsymbol{l} = (\delta l_x, \delta l_y, \delta l_z)$ according to the rules given in Algorithm 3.1.
- $\forall i \in \{x, y, z\}$: set $\delta p_i = 0$ if $\delta l_i = 1$ and $\delta p_i = 1$ otherwise.
- Refine the element $(K,\ \boldsymbol{p}_K)$ with the refinement indicators $\delta\boldsymbol{l}$ and $\delta\,\boldsymbol{p}$.

Algorithm 3.2: Geometric, anisotropic hp-mesh-degree refinement in three dimensions. Given a hexahedral mesh $\mathcal{T}$ of a bounded polyhedron D, a marking of singular vertices, edges and faces and a degree vector $\boldsymbol{p}$, this algorithm applies one anisotropic refinement step and returns the resulting mesh-degree combination $(\mathcal{T}',\ \boldsymbol{p}')$.

- Create an initial regular hexahedral mesh $\mathcal{T}^1$ of D and choose the initial degree vector $\boldsymbol{p}^1 = \left\{\boldsymbol{p}_K = (1, 1, 1)\right\}_{K\in\mathcal{T}}$. This mesh has one layer.
- Mark vertices, edge and faces as 'singular' towards which the mesh-degree combination should be refined.
- Call Algorithm 3.2 $n - 1$ times.

Algorithm 3.3: Create an hp-mesh-degree combination $(\mathcal{T}^n,\ \boldsymbol{p}^n)$ with n layers, degree distribution parameter $m = 1$ and geometric grading parameter $\sigma = 1/2$ in any bounded polyhedron $D \subset \mathbb{R}^3$.

Values of 0 in any component of $\delta\,\boldsymbol{p}$ mean 'no change'. A value of 1 in δp_x means that the x component of $\boldsymbol{p}_K$ should be increased by 1 and so on. The polynomial degree distribution parameter is $m = 1$: In an isotropic mesh, the polynomial degree $\boldsymbol{p}_K$ of an element K on layer k is $\boldsymbol{p}_K = \max\{(1, 1, 1), (\lceil mk\rceil, \lceil mk\rceil, \lceil mk\rceil)\}$.

Remark 3.8 (Simultaneous change of mesh and polynomial degree)
Note that Algorithm 3.2 simultaniously refines the geometric mesh $\mathcal{T}$ and the polynomial degrees $\boldsymbol{p}$ contrary to p-FEM software like StressCheck [69] or Nεκταr [72]. These software packages take an initial mesh and do not change it but only increase the polynomial degrees.

The meshes presented in Section 3.1.2 (geometric vertex, edge and boundary layer meshes) with geometric grading parameter $\sigma = 1/2$ and n layers can be generated using Algorithm 3.3. To create a geometric boundary layer mesh, the faces

with boundary layers have to be marked as 'singular'. Similarly for a geometric vertex (edge) mesh: there, the singular vertices (edges) have to be marked. Arbitrary combinations of geometric vertex, edge and boundary layer meshes are possible by marking the respective vertices, edges and faces. There is no restriction that they have to be on the boundary of the domain D: geometric meshes towards internal singularities or internal layers are also possible.

Finally, an hp-FE space is created using the mesh-degree combination $(\mathcal{T}^n, \boldsymbol{p}^n)$ with n layers from Algorithm 3.3: $V_N = \mathcal{S}^{\boldsymbol{p}^n,1}_{\Gamma_D}(D, \mathcal{T}^n)$.

Remark 3.9 (Sequence of hierarchic hp-FE spaces)

- *Algorithm 3.3 creates a sequence of mesh-degree combinations* $\{(\mathcal{T}^n, \boldsymbol{p}^n)\}_n$ *by calling Algorithm 3.2 several times. It is straight forward to generate a sequence of hp-FE spaces* $\{V_N\}_N$ *from* $\{(\mathcal{T}^n, \boldsymbol{p}^n)\}_n$.
- *The hp-FE spaces generated by Algorithm 3.3 are hierarchic, i. e. let* $V_{N_1} = \mathcal{S}^{\boldsymbol{p}^n,1}_{\Gamma_D}(D, \mathcal{T}^n)$ *and* $V_{N_2} = \mathcal{S}^{\boldsymbol{p}^{n+k},1}_{\Gamma_D}(D, \mathcal{T}^{n+k})$ *be the resulting spaces generated by Algorithm 3.3 with n and* $n+k$ *layers respectively. Then,* $V_{N_1} \subseteq V_{N_2}$ *for any* $k \geq 0$.

(2.50) is a key property of the determinant of the Jacobian of the cells in the corner-edge neighbourhood $\mathcal{V}_e(\boldsymbol{a})$ including edge. It is verified in the result below.

Proposition 3.10 *Given any bounded polyhedron* $D \in \mathbb{R}^3$, *Algorithm 3.3 generates a mesh* $\mathcal{T}^n$ *fulfilling* (2.50).

Proof: The condition on the Jacobian determinant for cells in the corner-edge neighbourhood $\mathcal{V}_e(\boldsymbol{a})$ including edge (2.50) is

$$C_1 h^3 s^2 \leq \det \frac{dF_K}{d\boldsymbol{\xi}} \leq C_2 h^3 s^2,$$

where

$$h := \overline{\varkappa}_1(K) = \max_{\boldsymbol{x} \in K} r_{\boldsymbol{a}}(\boldsymbol{x}),$$

$$s := \overline{\varkappa}_2(K) = \max_{\boldsymbol{x} \in K} \sin \varphi(\boldsymbol{x}) = \max_{\boldsymbol{x} \in K} \frac{r_e(\boldsymbol{x})}{r_{\boldsymbol{a}}(\boldsymbol{x})}.$$

The element map $F_K : \hat{K} \to K$ of an element $K \in \mathcal{T}^n$ is a composition of a trilinear element map $F_{\tilde{K}} : \hat{K} \to \tilde{K}$ with $K \subset \tilde{K} \in \mathcal{T}^1$ and an affine element map $F_{K^{\text{aff}}}$:

$$F_K = F_{\tilde{K}} \circ F_{K^{\text{aff}}},$$

where the affine map $F_{K^{\text{aff}}}$ describes the subdivision in the reference situation, *i. e.* it maps the reference element of K to a child of the reference element of $\tilde{K}$.

The determinant of the Jacobian of F_K is

$$\det \frac{dF_K}{d\xi} = \det \left[\frac{dF_{\tilde{K}}}{d\xi} \circ F_{K^{\text{aff}}} \cdot \frac{dF_{K^{\text{aff}}}}{d\xi} \right] = \det \left[\frac{dF_{\tilde{K}}}{d\xi} \circ F_{K^{\text{aff}}} \right] \cdot \det \frac{dF_{K^{\text{aff}}}}{d\xi}.$$

The factor $\det \left[\frac{dF_{\tilde{K}}}{d\xi} \circ F_{K^{\text{aff}}} \right]$ can be bounded from above and below independently of the number of layers n in $\mathcal{T}^n$ only depending on the initial hexahedral mesh $\mathcal{T}^1$. Likewise,

$$\tilde{c}_1 \|F_{\tilde{K}}\| \|F_{K^{\text{aff}}}(\xi)\| \leq \|F_{\tilde{K}} \circ F_{K^{\text{aff}}}(\xi)\| \leq \tilde{c}_2 \|F_{\tilde{K}}\| \|F_{K^{\text{aff}}}(\xi)\|.$$

Therefore, it suffices to study

$$C_1^{\text{aff}} h^3 s^2 \leq \det \frac{dF_{K^{\text{aff}}}}{d\boldsymbol{\xi}} \leq C_2^{\text{aff}} h^3 s^2, \tag{3.11}$$

i. e. we are in the reference situation (here, h and s are also taken in the reference situation). The condition (3.11) needs to be verified for the cells in $\mathcal{V}_e(\boldsymbol{a})$, *e. g.* the grayed cells in Figure 3.7.

The *layers* are denoted with $l_1 = 0, 1, \ldots$ perpendicular to the edge e and the *levels* are denoted with $l_2 = 1, 2, \ldots$ along the edge. Figure 3.7 shows the layers $l_1 = 0, 1, 2$ and the levels $l_2 = 1, 2, 3$. In a mesh with n layers (Figure 3.7 has 5 layers),

$$\det \frac{dF_{K^{\text{aff}}}}{d\boldsymbol{\xi}} = 2^{3(n-1)+l_2+2\max\{0,l_1-1\}}.$$

To estimate the upper and lower bounds in (3.11), upper and lower bounds for h and s in each layer-level combination (l_1, l_2) are required:

$$\begin{aligned}
r_{e,\text{up}}^2 &:= 2 \cdot 2^{2(-n+1+l_1)}, & r_{e,\text{low}}^2 &:= 2^{2(-n+1+l_1)} + 2^{2(-n+l_1)}, \\
h_{\text{up}}^2 &= 2^{2(-n+2+l_2)} + r_{e,\text{up}}^2, & h_{\text{low}}^2 &= 2^{2(-n+2+l_2)} + r_{e,\text{low}}^2, \\
s_{\text{up}}^2 &= \frac{r_{e,\text{up}}^2}{2^{2(-n+1+l_2)} + r_{e,\text{up}}^2}, & s_{\text{low}}^2 &= \frac{r_{e,\text{low}}^2}{2^{2(-n+1+l_2)} + r_{e,\text{low}}^2}.
\end{aligned} \tag{3.12}$$

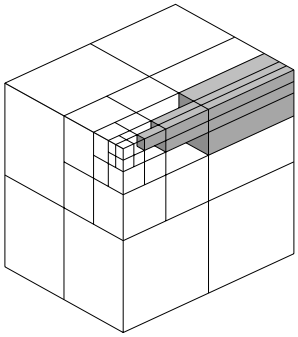


Figure 3.7: Cells in the corner-edge neighbourhood $\mathcal{V}_e(\boldsymbol{a})$ including edge. This mesh has 5 layers.

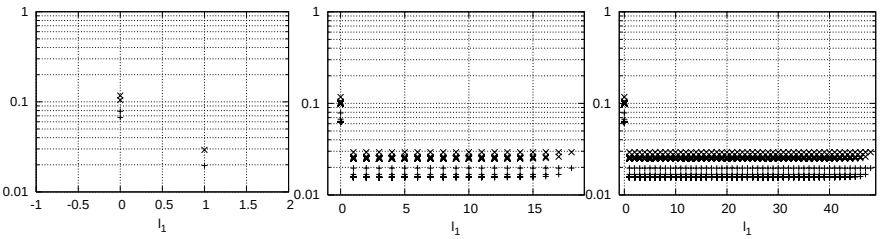


Figure 3.8: Plot of (3.13) for each cell in $\mathcal{V}_e(\boldsymbol{a})$ in $\mathcal{T}^n$ against l_1 for $n = 4$, 21 and 51 (from left to right). $\times$ and $+$ are computed with h_{up}, s_{up} and h_{low}, s_{low} respectively.

Figure 3.8 shows plots of the quotient

$$\frac{\det \frac{dF_{K^{\text{aff}}}}{d\boldsymbol{\xi}}}{h^3 s^2} \tag{3.13}$$

for different number of layers n for the upper and lower bounds of h and s. Clearly, the quotient is bounded from above and below. It is also possible to compute the bounds manually. □

Theorem 3.14 *Given any bounded polyhedron $D \in \mathbb{R}^3$, Algorithm 3.3 generates a mesh-degree combination $(\mathcal{T}^n, \boldsymbol{p}^n)$ fulfilling Definition 2.67 with degree distri-*

bution parameter $m = 1$ *and grading factor* $\sigma = 1/2$. $\mathcal{T}^n$ *consists of trilinearly mapped hexahedra.*

Proof: The proof consists of three parts. First, the initial mesh-degree combination $(\mathcal{T}^1, \boldsymbol{p}^1)$ is specified. Then, the second and third part inductively deal with the degrees and sizes of the elements in an invocation of Algorithm 3.2.

1. As shown in Proposition 3.5, there exists a shape-regular mesh consisting of trilinearly mapped hexahedra in any bounded polyhedron in $\mathbb{R}^3$. The initial coarse grid $(\mathcal{T}^1, \boldsymbol{p}^1)$ generated in the first step of Algorithm 3.3 fulfils Definition 2.67.

2. Applying Algorithm 3.2 to a degree vector $\boldsymbol{p}$ (induction over the number of mesh layers):

   - The vertex cell K is newly created. Therefore, its degree $\boldsymbol{p}_K$ is not increased, *i. e.* it stays at $\boldsymbol{p}_K = (1, 1, 1)$.
   - The degree $\boldsymbol{p}_K$ of a cell in the *regular part* $\mathcal{V}^0$ is increased isotropically.
   - The degrees $\boldsymbol{p}_K$ of the cells K in the *corner neighbourhood* $\mathcal{V}^0_{\boldsymbol{a}}$ *excluding corner-edge neighbourhoods* on layer $\mathcal{L}_k$, $k \geq 1$ are increased isotropically as all elements not touching a singular edge or vertex are treated the same as an element in the regular part. The elements on the layers $\mathcal{L}_k$, $k = 0, 1$, are created in this invocation of the algorithm. The elements on $\mathcal{L}_0$ are vertex elements, *i. e.* $\boldsymbol{p}_K = (1, 1, 1)$, and those on $\mathcal{L}_1$ also have $\boldsymbol{p}_K = (1, 1, 1)$.
   - The degrees $\boldsymbol{p}_K$ of the cells in the *edge neighbourhood* $\mathcal{V}^0_e$ *excluding corner* are increased isotropically as long as they do not touch the edge. For these cells, the same arguments as in the corner neighbourhood $\mathcal{V}^0_{\boldsymbol{a}}$ excluding corner-edge neighbourhoods holds. If the cell touches the edge e, it is split up anisotropically along the edge and its degree is increased in the direction along the edge. Assuming this is the n-th invocation of Algorithm 3.2 and e is parallel to the z axis (in local coordinates of the cell), the degree of these newly created cells is $\boldsymbol{p}_K = (1, 1, n + 1)$.
   - Combining the arguments for $\mathcal{V}^0_e$ and $\mathcal{V}^0_{\boldsymbol{a}}$ shows that the degrees $\boldsymbol{p}_K$ of the cells in the *corner-edge neighbourhood* $\mathcal{V}_e(\boldsymbol{a})$ *including edge* behave as required.

3. Denote by $\mathcal{T} = \{K\}$ and $\mathcal{T}' = \{K'\}$ the old and new mesh respectively. Let n be the number of layers in $\mathcal{T}$. Applying Algorithm 3.2 to $\mathcal{T}$ only changes the cells K touching a singular corner or edge. Hence, $\mathcal{T}'$ has $n+1$ layers.

   - For a *vertex cell* $K'_{\boldsymbol{a}}$, $\operatorname{diam} K'_{\boldsymbol{a}} = 1/2 \operatorname{diam} K_{\boldsymbol{a}}$ holds. Therefore, (2.65) holds:
   $$\underline{C}\sigma^{n+1} \leq \operatorname{diam}(K'_{\boldsymbol{a}}) \leq \overline{C}\sigma^{n+1}.$$

   - Most cells K in the *corner neighbourhood* $\mathcal{V}^0_{\boldsymbol{a}}$ *excluding corner-edge neighbourhoods* are not changed. However, the old vertex cell $K_{\boldsymbol{a}}$ is split up. Therefore, (2.64) holds with
   $$\min_{K' \subset \mathcal{V}^0_{\boldsymbol{a}}} \overline{\varkappa}(K') = 1/2 \min_{K \subset \mathcal{V}^0_{\boldsymbol{a}}} \overline{\varkappa}(K).$$

   - As before, the cell K'_e in the terminal layer of the *edge neighbourhood* $\mathcal{V}^0_e$ *excluding corner* fulfils (2.62) with
   $$\overline{\varkappa}(K'_e) = 1/2\, \overline{\varkappa}(K_e).$$

   - The second part of (2.63) is asserted by the same argument as for $\mathcal{V}^0_{\boldsymbol{a}}$. The first part of (2.63) for the cells in the terminal layer of the *corner-edge neighbourhood* $\mathcal{V}_e(\boldsymbol{a})$ *including edge:*
   $$\overline{\varkappa}_2(K'_e) \leq \overline{C}\sigma^n$$
   is fulfilled by $\overline{\varkappa}_2(K'_e) = 1/2\, \overline{\varkappa}_2(K_e)$ for sufficiently small cells K_e.

Summarising steps 2 and 3, an old mesh-degree combination $(\mathcal{T}, \boldsymbol{p})$ fulfilling Definition 2.67 is transformed into another valid mesh-degree combination $(\mathcal{T}', \boldsymbol{p}')$. Together with the fact that the coarse grid $(\mathcal{T}^1, \boldsymbol{p}^1)$ also fulfils Definition 2.67 (step 1), this concludes the proof. □

Proposition 3.15 (Complexity of Algorithm 3.3) *The run-time complexity of Algorithm 3.3 is*
$$rt_2(n) = \mathcal{O}(n^2),$$
i. e. the complexity for generating the hp-mesh-degree combination $(\mathcal{T}^n, \boldsymbol{p}^n)$ *grows quadratically in the number of mesh layers n.*

Proof: Obviously, Algorithms 3.1 and 3.2 have run-time complexity $\mathcal{O}(\#\mathcal{T})$, where $\#\mathcal{T}$ is the number of cells in the mesh $\mathcal{T}$. The number of cells per layer is bounded by L_0 independently of n [63]. Therefore, the run-time complexity of the Algorithms 3.1 and 3.2 on the mesh $\mathcal{T}^i$ is $rt_1(i) = \mathcal{O}(i)$.

It takes $n-1$ invocations of Algorithm 3.2 to generate a mesh with n layers:

$$rt_2(n) = \sum_{i=1}^{n-1} rt_1(i) = \mathcal{O}\left(1/2(n-1)(n-2)\right) = \mathcal{O}(n^2).$$

□

3.3 Generic Assembly Procedure

A general variational formulation for FEM reads:
Find $u \in V$ such that

$$a(u, v) = l(v) \quad \forall v \in V. \tag{3.16}$$

To approximate $u \in V$ numerically, recall that a discretisation of the space V is defined using a finite dimensional set of linearly independent functions $\{\Phi_i\}$ forming a basis of $V_N \subset V$. The bilinear form $a(.,.)$ and linear form $l(.)$ are used to generate the stiffness matrix $\underline{A}$ and load vector $\boldsymbol{l}$ on the space V_N. Finally, the linear system $\underline{A}^\top \boldsymbol{u} = \boldsymbol{l}$ is solved for $\boldsymbol{u}$, $u_N = \boldsymbol{\Phi}^\top \boldsymbol{u}$ an approximation for u.

Algorithmically, $\underline{A}$ and $\boldsymbol{l}$ are built element-wise, *i. e.* a triangulation of D into a mesh $\mathcal{T} = \{K\}$ is used. On these cells K, the local shape functions are defined according to the degree $\boldsymbol{p}_K$. The shape functions are assembled by T matrices into the global basis functions.

3.3.1 T Matrices

Definition 3.17 (T matrix) *Let m_K the number of local shape functions $\{\varphi_j^K\}_{j=1}^{m_K}$ in the element K and N the number of global basis functions $\{\Phi_i\}_{i=1}^N$. The T matrix $\underline{T}_K \in \mathbb{R}^{m_K \times N}$ of the element K describes how the restriction of the global basis functions $\{\Phi_i\}_{i=1}^N$ onto the element K are constructed from the local shape functions:*

$$\Phi_i|_K = \sum_{j=1}^{m_K} \left[\underline{T}_K\right]_{ji} \varphi_j^K$$

and in vector notation: $\boldsymbol{\Phi}|_K = \underline{T}_K^\top \boldsymbol{\varphi}^K$.

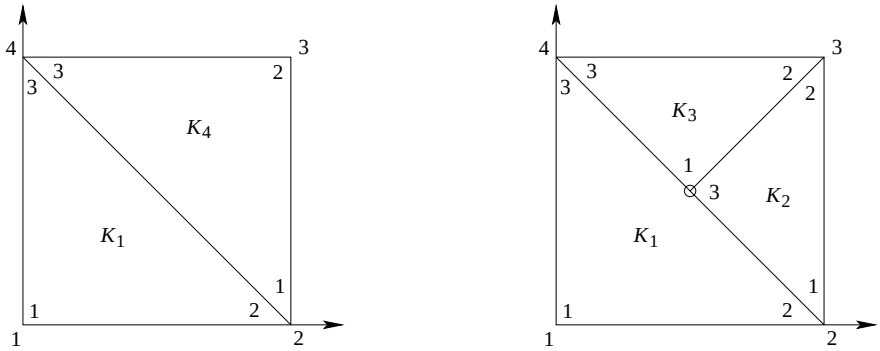


Figure 3.9: Regular (left) and irregular (right) mesh with two elements with three local shape functions (linear) each and four global basis functions (hat functions). The hanging node in the irregular mesh is marked with a ∘.

In classical FEM, the basis functions have to be continuous. From the point of view of the element shape functions, the T matrices ensure the continuity of the global basis functions by assembling them correctly. In others methods like the discontinuous Galerkin FEM (DGFEM) and Boundary Element Method (BEM) (*c. f.* Sections 2.4.1 and 2.4.2 respectively), the basis functions are usually discontinuous. This can also be described by means of T matrices.

Example 3.18 (T matrix of a regular mesh) *In a regular mesh*[2]*, there is a "one-to-one" correspondence of local shape functions and global basis functions: Every local shape function contributes to at most one global basis function. On the other hand, the restriction of a global basis function onto an element gets contributions from at most one local shape function.*

Consider the regular mesh shown in Figure 3.9 on the left. Assume standard linear nodal shape functions on the elements K_1 and K_4 forming the usual hat functions. The elements K_1 and K_4 have the T matrices

$$\underline{T}_{K_1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \text{ and } \underline{T}_{K_4} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{3.19}$$

[2]The intersection of any two elements is either empty, a vertex or a side, *c. f.* Figure2.1 on page 28.

The row and column indices in the T matrices in (3.19) *correspond to the indices of the local shape functions (numbers inside the triangles in Figure 3.9) and the global basis functions (numbers outside the triangles in Figure 3.9) respectively.*

As a result of the "one-to-one" correspondence of local shape functions and global basis functions, every row and every column of the two T matrices in (3.19) *have at most one entry equal to 1.*

Example 3.20 (T matrix of an irregular mesh) *Consider the mesh shown on the right of Figure 3.9. The mesh is irregular. Clearly, there is no such "one-to-one" correspondence of local shape functions and global basis function as it was the case for the regular mesh.*

Using Definition 3.17, the T matrices of the elements K_2 and K_3 are:

$$\underline{T}_{K_2} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1/2 & 0 & 1/2 \end{pmatrix} \text{ and } \underline{T}_{K_3} = \begin{pmatrix} 0 & 1/2 & 0 & 1/2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{3.21}$$

3.3.2 Assembly Procedure

Restricting the bilinear form $a(.,.)$ and the linear form $l(.)$ to an element K makes it possible to compute the element stiffness matrices and element load vectors. The global stiffness matrix $\underline{A}$ is assembled from the element stiffness matrices using the T matrices:

$$\begin{aligned} \underline{A} = a_D(\boldsymbol{\Phi}, \boldsymbol{\Phi}) &= a\Big(\sum_K \underline{T}_K^\top \boldsymbol{\varphi}^K, \sum_{\tilde{K}} \underline{T}_{\tilde{K}}^\top \boldsymbol{\varphi}^{\tilde{K}}\Big) \\ &= \sum_{K,\tilde{K}} \underline{T}_K^\top a_{K\cup\tilde{K}}(\boldsymbol{\varphi}^K, \boldsymbol{\varphi}^{\tilde{K}}) \underline{T}_{\tilde{K}} = \sum_{K,\tilde{K}} \underline{T}_K^\top \underline{A}_{K\tilde{K}} \underline{T}_{\tilde{K}}. \end{aligned}$$

Here, $a_D(.,.)$ denotes the bilinear form $a(.,.)$ from (3.16) and $a_{K\cup\tilde{K}}(.,.)$ the bilinear form over the domain $K \cup \tilde{K}$ taking as input local shape functions φ_j. $a_{K\cup\tilde{K}}(.,.)$ is evaluated by extending the φ_j with 0 to D and using $a(.,.)$.

This leads to the formal definition of the assembly operator:

Definition 3.22 (Assembly operator) *The assembly operator $\mathcal{A}$ is defined as follows:*

- *for assembling matrices*

$$\underline{A} = \mathcal{A}_{K,\tilde{K}\in\mathcal{T}}\,\underline{A}_{K\tilde{K}} := \sum_{K,\tilde{K}\in\mathcal{T}} \underline{T}_K^\top \underline{A}_{K\tilde{K}} \underline{T}_{\tilde{K}}, \tag{3.23}$$

- *for assembling vectors*

$$\boldsymbol{l} = \mathcal{A}_{K\in\mathcal{T}}\,\boldsymbol{l}_K := \sum_{K\in\mathcal{T}} \underline{T}_K^\top \boldsymbol{l}_K. \tag{3.24}$$

Remark 3.25 *In classical FEM, the double sum in* (3.23) *collapses into a single sum since* $\underline{A}_{K\tilde{K}} \equiv 0$ *for* $K \neq \tilde{K}$*. This is not the case for non-local operators as e. g. those in the Boundary Element Method or in the discontinuous Galerkin FEM.*

This idea of T matrices separates the computation of the local element matrices from the global stiffness matrix and the assembly process. The latter two can stay unchanged even if new global basis functions, local shape functions or physical problems (involving new element matrices) are introduced. This clear separation on the algorithmic side allows clean interfaces on the implementation side and the above mentioned properties carry over to the implementation (*c. f.* Section 6.1.2).

3.3.3 Generation of T Matrices

The generation of a T matrix for a regular mesh depends heavily on counting and assigning indices with respect to topological entities such as vertices, edges and faces. For an irregular mesh, this could also be done in a similar way. The present and the following section describe a way to simplify the generation of the T matrices for the cells in an irregular part of the mesh.

Regular Meshes

Generating a T matrix for a regular mesh is just a matter of counting and assigning global and local degrees of freedom. A global degree of freedom corresponds to a global basis function and a local degree of freedom corresponds to a local shape function. All local degrees of freedom are associated to a topological entity of the mesh (like a vertex, an edge, a face or a cell).

An element's T matrix is generated column by column: Every column holds a global to local degree of freedom association. The column index is the number of the global degree of freedom and the row index is the number of the local degree of freedom. In Example 3.18, the T matrices of the elements K_1 and K_4 both have four columns. Every column is associated to a global degree of freedom (see Figure 3.9 for the degrees of freedom)—every row is associated to a local degree of freedom.

Irregular Meshes

Irregularities in meshes (*c. f.* Figure 3.9) can arise from different sources: overlapping meshes, moving parts of a mesh or local refinements. We consider meshes with hanging nodes (like to ones in Figures 3.9, 3.10 and 3.12) which result from (possibly recursive) local refinements of a initially consistent mesh: the mesh on the right of Figure 3.9 is a refined version of the initially regular mesh on the left.

In many other algorithms and implementations, several situations such as singly constrained nodes (also referred to as 1-irregular meshes) or doubly constrained nodes have to be distinguished and treated separately.[3] In the following, we present an approach that extends the simple construction rules for regular meshes to the irregular case and is not restricted to 1-irregular meshes. Briefly, if an element has hanging nodes, the T matrix for the regular parent is generated and modified by an S matrix (defined below).

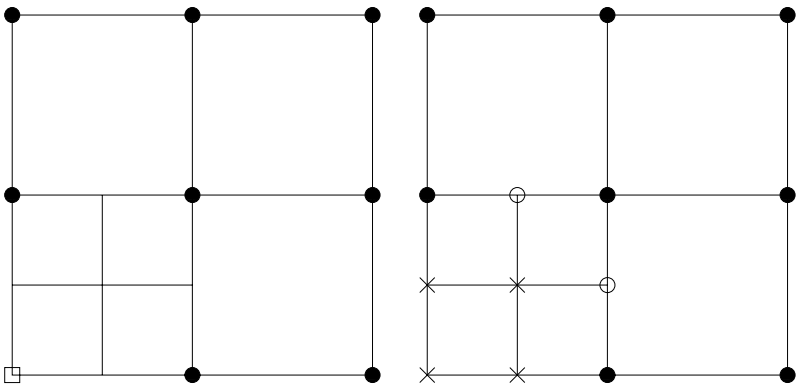
Consider a mesh $\mathcal{T}$ for which all elements and associated T matrices have been generated. Suppose the mesh $\mathcal{T}'$ is the result of splitting several elements of $\mathcal{T}$. The basis functions $B := \{\Phi_1, \ldots, \Phi_N\}$ defined for $\mathcal{T}$ may be partitioned into two sets—one denoted by B_{replace} containing all basis functions that can be described solely by elements of $\mathcal{T}'$ that are not part of $\mathcal{T}$ and another one denoted by B_{keep} representing the rest:

$$B = B_{\text{replace}} \cup B_{\text{keep}}.$$

Note that B_{replace} is easily determined; the support of basis functions in B_{replace} consists entirely of newly inserted elements.

The set of basis functions B' related to the mesh $\mathcal{T}'$ contains all basis functions in B_{keep} plus an additional set B_{insert} of basis functions generated by regular

[3] There exist many implementations of this so-called *constrained approximation*, many of them with different ideas and algorithms [14, 15, 41, 42, 43, 47, 50, 68].



(a) Initial mesh $\mathcal{T}$ with indicated refinement. Basis functions B located at $\bullet$ and $\square$: $\bullet \in B_{\text{keep}}$ and $\square \in B_{\text{replace}}$.

(b) Refined mesh $\mathcal{T}'$ with hanging nodes $\circ$, $\bullet \in B_{\text{keep}}$, $\times \in B_{\text{insert}}$.

Figure 3.10: Refinement of the lower left quadrilateral of a mesh $\mathcal{T}$ consisting of four quadrilaterals (a) into the new mesh $\mathcal{T}'$ consisting of seven quadrilaterals (b). Indicated are the elements of $B = B_{\text{replace}} \cup B_{\text{keep}}$ in (a) and $B' = B_{\text{insert}} \cup B_{\text{keep}}$ in (b).

components of mesh $\mathcal{T}'$ formed by elements not part of $\mathcal{T}$:

$$B' = B_{\text{insert}} \cup B_{\text{keep}}.$$

The remaining basis functions in B_{keep} are not modified although this could increase the efficiency. On the other hand, we are able to exploit the tensor product structure (*c. f.* Section 3.3.5).

Example 3.26 *In Example 3.18 we find $B = B'$ since $B_{replace} = \emptyset = B_{insert}$.*

Example 3.27 *Consider the mesh in Figure 3.10 resulting from refining the lower left quadrilateral. Assume bilinear nodal basis functions indicated by different symbols in the vertices of the mesh.*

The mesh $\mathcal{T}$ is the one containing the four quadrilaterals and B consist of the nine hat functions indicated by $B_{keep} = \{\bullet\}$ and $B_{replace} = \{\square\}$ in Figure 3.10(a).

Figure 3.10(b) shows how B' is split into $B_{keep} = \{\bullet\}$ and $B_{insert} = \{\times\}$. $\circ$ indicate hanging nodes.

Every element of a set of basis functions B or B' has a column in the T matrix of every element. The entries in such a column give the coefficients of the respective local shape functions with respect to the chosen global basis function. As mentioned above, generating a T matrix for a regular mesh is just a matter of counting and assigning indices to topological entities. This holds for $B_{\text{insert}} \subset B'$ since B_{insert} consist of the basis functions in the regular part of $\mathcal{T}'$. The columns in the T matrices of the basis functions in B_{keep} have to be modified by a so called S matrix.

3.3.4 S Matrices

An S matrix follows the same idea as a T matrix: it describes how the larger functions are constructed from the smaller ones.

Definition 3.28 (S matrix) *Let $K' \subset K$ be the result of an h-refinement of element K. The S matrix $\underline{S}_{K'K} \in \mathbb{R}^{m_{K'} \times m_K}$ describes how the restriction of the shape functions $\{\varphi_j^K\}_{j=1}^{m_K}$ onto K' are constructed from the shape functions $\{\varphi_l^{K'}\}_{l=1}^{m_{K'}}$ of K':*

$$\varphi_j^K\Big|_{K'} = \sum_{l=1}^{m_{K'}} \left[\underline{S}_{K'K}\right]_{lj} \varphi_l^{K'}$$

and in vector notation: $\boldsymbol{\varphi}^K\big|_{K'} = \underline{S}_{K'K}^\top \boldsymbol{\varphi}^{K'}$. In the trivial case $K = K'$ (i. e. no refinement), the S matrix $\underline{S}_{KK}$ is equal to the identity matrix.

Similar ideas can be found in the construction of prolongation operators in multigrid methods.

Proposition 3.29 (Application of S matrix) *Let $K' \subset K$ be the result of a refinement of an element K. Then, the T matrix of K' can be computed as*

$$\underline{T}_{K'} = \underline{S}_{K'K}\tilde{\underline{T}}_K + \tilde{\underline{T}}_{K'},$$

where $\tilde{\underline{T}}_K$ denotes the T matrix of element K with columns not related to functions in B_{keep} set to zero and $\tilde{\underline{T}}_{K'}$ the T matrix for functions in B_{insert} with respect to K'.

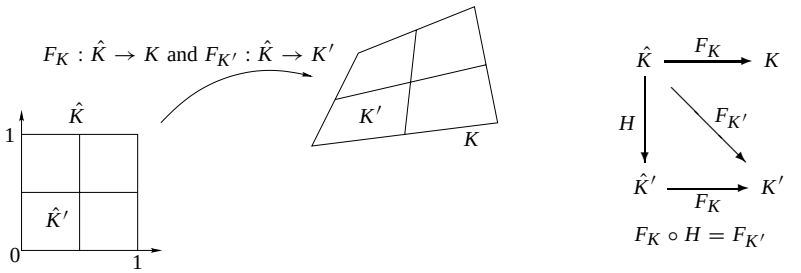


Figure 3.11: Element maps for K and K'.

Proof: If $K = K'$, nothing is to be proved. Let now $K' \subset K$ with strict inclusion. Consider the global basis function Φ_i restricted to the element K':

$$\Phi_i|_{K'} = \Phi_i|_K|_{K'} = \sum_{j=1}^{m_K} \left[\underline{T}_K\right]_{ji} \varphi_j^K\Big|_{K'} = \sum_{j=1}^{m_K} \left[\underline{T}_K\right]_{ji} \sum_{l=1}^{m_{K'}} \left[\underline{S}_{K'K}\right]_{lj} \varphi_l^{K'},$$

i. e. $\left[\underline{T}_{K'}\right]_i = \underline{S}_{K'K} \left[\underline{T}_K\right]_i$ for all $\Phi_i \in B_{\text{keep}}$. For $\Phi_i \in B_{\text{insert}}$, the assertion holds by definition. □

The S matrices do not depend on the exact geometry of the elements but only on the topology and the subdivision ratio.

Proposition 3.30 (S matrix in reference situation) *Let $\hat{K}' \subset \hat{K}$ be the result of a refinement of the reference element $\hat{K}$ with $H : \hat{K} \to \hat{K}'$ the subdivision map (see Figure 3.11). The element maps are $F_K : \hat{K} \to K$ and $F_{K'} : \hat{K} \to K'$ and*

$$F_{K'} \circ H^{-1} = F_K \tag{3.31}$$

holds. Then, $\underline{S}_{\hat{K}'\hat{K}} = \underline{S}_{K'K}$.

Proof: The local element shape functions and the reference element shape functions are connected by the element map:

$$\begin{aligned} \varphi_j^K \circ F_K &= N_j, \\ \varphi_j^{K'} \circ F_{K'} &= N_j. \end{aligned} \tag{3.32}$$

Using Definition 3.28:

$$\varphi_j^K\Big|_{K'} = \sum_{l=1}^{m_{K'}} \left[\underline{S}_{K'K}\right]_{lj} \varphi_l^{K'}.$$

Taking the local element shape functions back to the reference element $\hat{K}$ by F_K yields

$$\varphi_j^K \circ F_K\Big|_{\hat{K}'} = \sum_{l=1}^{m_{K'}} \left[\underline{S}_{K'K}\right]_{lj} \varphi_l^{K'} \circ F_K.$$

Using (3.31) and (3.32), it follows:

$$N_j\big|_{\hat{K}'} = \sum_{l=1}^{m_{K'}} \left[\underline{S}_{K'K}\right]_{lj} N_l \circ H^{-1}. \tag{3.33}$$

Comparing (3.33) and the definition of the S matrix $\underline{S}_{\hat{K}'\hat{K}}$

$$N_j\big|_{\hat{K}'} = \sum_{l=1}^{m_{\hat{K}'}} \left[\underline{S}_{\hat{K}'\hat{K}}\right]_{lj} N_l \circ H^{-1}$$

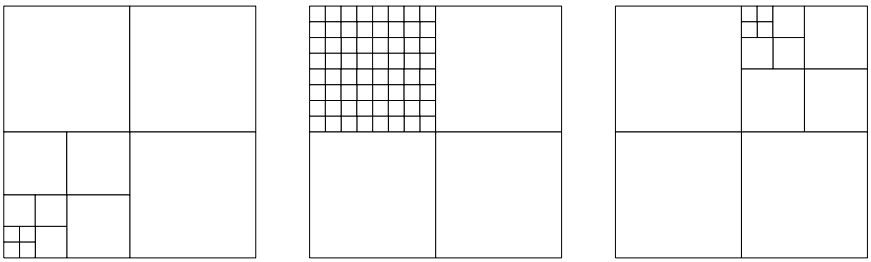
concludes the proof. □

Example 3.34 (Examples of Meshes) *The meshes in Figure 3.12 can be handled by the algorithms with T and S matrices presented in this section. The subdivision ratio used here is $\sigma = 1/2$, i. e. the children all have the same size.*

Mesh (a) is constructed by a geometric refinement towards the lower left corner. This mesh is 1-irregular and could also be handled by an algorithm which is able to eliminate only singly constrained nodes.

Mesh (b) shows a quadrilateral (the top left one) which is refined three times recursively, i. e. it is divided into $2^3 \cdot 2^3 = 64$ small quadrilaterals. This can no longer be handled by an algorithm which is able to handle 1-irregular meshes only. An S matrix can be applied recursively as it was done for the subdivision algorithm, though.

The mesh (c) does not look very different to mesh (a) but the constrained nodes are not just singly constrained. Again, the S matrix has to be applied several times.



(a) 1-irregular mesh, geometric refinement towards lower left corner.

(b) 3 uniform refinements of top left quadrilateral, not 1-irregular.

(c) Geometric refinement of the top right quadrilateral towards its top left corner. Not 1-irregular.

Figure 3.12: Different meshes which can be handled by applying the S matrices recursively.

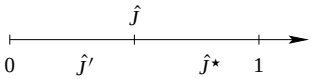


Figure 3.13: One dimensional reference element with left and right child.

3.3.5 Generation of S Matrices

If, in higher dimensions, the reference element shape functions are tensorised one dimensional reference element shape functions, the S matrices also have a tensor product structure. Therefore, we only consider reference elements which allow tensorised element shape functions: quadrilaterals and hexahedra in two and three dimensions, respectively.

For other element types like triangles or tetrahedra, the S matrices need to be computed directly by solving a linear system as it is done in one dimension (see below).

S Matrix in One Dimension

Let the reference element shape functions $\{N_j\}$ be defined on $\hat{J} = (0,1)$ (*c.f.* Figure 3.13). In one dimension, the S matrices can be computed by solving a linear system. Given the subdivision map $G : \hat{J} \to \hat{J}'$, $\xi \mapsto \xi/2$, there holds

$$\boldsymbol{N}|_{\hat{J}'} = \underline{S}^{\top}_{\hat{J}'\hat{J}} \boldsymbol{N} \circ G^{-1}, \tag{3.35}$$

Evaluating (3.35) in $m_{\hat{J}}$ distinct points in the interval $[0, 1/2]$ results in a linear system which can be solved for $\underline{S}_{\hat{J}'\hat{J}}$. The same holds for $\underline{S}_{\hat{J}^\star\hat{J}}$.

Example 3.36 *For the reference element shape functions (c.f. Section 6.3.2)*

$$N_j(\xi) = \begin{cases} 1-\xi & j = 1 \\ \xi & j = 2 \\ \xi(1-\xi)P^{1,1}_{j-3}(2\xi-1) & j = 3, \ldots, m_{\hat{J}} \end{cases} \tag{3.37}$$

the S matrices are ($m_{\hat{J}} = 4$):

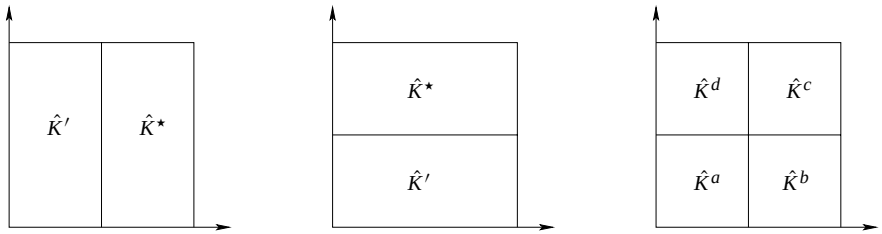
$$\underline{S}_{\hat{J}'\hat{J}} = \begin{pmatrix} 1/2 & 1/2 & 1/4 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1/4 & 3/4 \\ 0 & 0 & 0 & 1/8 \end{pmatrix} \text{ and } \underline{S}_{\hat{J}^\star\hat{J}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1/2 & 1/2 & 1/4 & 0 \\ 0 & 0 & 1/4 & -3/4 \\ 0 & 0 & 0 & 1/8 \end{pmatrix}.$$

Remark 3.38 *If the reference element shape functions are hierarchical—like those given in* (3.37)*—the S matrices are hierarchical too. Therefore, in a FEM code, it is only necessary to store the S matrices for the highest occurring polynomial degree. If only moderate polynomial degrees are used, a computation (and caching) of the S matrices only when needed is feasible.*

S Matrix in Two Dimensions

Only quadrilaterals shall be considered as they allow tensor product reference element shape functions. The three different subdivisions of a quadrilateral $(0,1)^2$ shown in Figure 3.14 are treated. The reference element shape functions are tensorised one dimensional shape functions:

$$N_{i,j} = N_i \otimes N_j. \tag{3.39}$$



(a) Vertical. (b) Horizontal. (c) Into four.

Figure 3.14: Variants of subdividing a quadrilateral.

Consider the vertical subdivision variant in Figure 3.14(a) with the subdivision map

$$H : \hat{K} \to \hat{K}', \boldsymbol{\xi} \mapsto \begin{pmatrix} \xi_1/2 \\ \xi_2 \end{pmatrix}$$

for the right child $\hat{K}'$. By Definition 3.28, the S matrix $\underline{S}_{\hat{K}'\hat{K}}$ is defined as

$$N_{i,j}\big|_{\hat{K}'} = \sum_{k,l} \left[\underline{S}_{\hat{K}'\hat{K}}\right]_{(k,l),(i,j)} N_{k,l} \circ H^{-1}.$$

Inserting (3.39) into this yields

$$\left(N_i \otimes N_j\right)\big|_{\hat{K}'} = \sum_{k,l} \left[\underline{S}_{\hat{K}'\hat{K}}\right]_{(k,l),(i,j)} (N_k \otimes N_l) \circ H^{-1}. \tag{3.40}$$

The S matrices for the one dimensional reference element shape functions used in (3.40) are

$$N_i\big|_{\hat{\jmath}'} = \sum_m \left[\underline{S}_{\hat{\jmath}'\hat{\jmath}}\right]_{mi} N_m \circ G^{-1} \qquad \text{for the } \xi_1 \text{ part and}$$

$$N_j = \sum_n [\mathbb{I}]_{nj} N_n \qquad \text{for the } \xi_2 \text{ part.}$$

Note that the ξ_2 part is not refined and therefore, the S matrix is the identity matrix

$\mathbb{I}$. Plugging this into the left hand side of (3.40) gives:

$$\begin{aligned}\big(N_i \otimes N_j\big)\big|_{\hat{K}'} = N_i|_{\hat{J}'} \otimes N_j &= \sum_{m,n} \Big(\big[\underline{S}_{\hat{J}'\hat{J}}\big]_{mi}\, N_m \circ G^{-1} \Big) \otimes \big([\mathbb{I}]_{nj}\, N_n \big) \\ &= \sum_{m,n} \big[\underline{S}_{\hat{J}'\hat{J}}\big]_{mi} \cdot [\mathbb{I}]_{nj}\, N_m \circ G^{-1} \otimes N_n\end{aligned}$$

comparing with the right hand side of (3.40)

$$= \sum_{k,l} \big[\underline{S}_{\hat{K}'\hat{K}}\big]_{(k,l),(i,j)}\, N_k \circ G^{-1} \otimes N_l$$

results in (using the definition of matrix tensor products of Fiedler [52]):

$$\underline{S}_{\hat{K}'\hat{K}} = \underline{S}_{\hat{J}'\hat{J}} \otimes \mathbb{I} \qquad \text{for the left quadrilateral } \hat{K}'.$$

The right child $\hat{K}^\star$ in the vertical subdivision variant has the S matrix

$$\underline{S}_{\hat{K}^\star\hat{K}} = \underline{S}_{\hat{J}^\star\hat{J}} \otimes \mathbb{I}.$$

Similarly, the children of the horizontal subdivision variant in Figure 3.14(b) have

$$\begin{aligned}\underline{S}_{\hat{K}'\hat{K}} &= \mathbb{I} \otimes \underline{S}_{\hat{J}'\hat{J}} && \text{for the bottom quadrilateral } \hat{K}' \text{ and} \\ \underline{S}_{\hat{K}^\star\hat{K}} &= \mathbb{I} \otimes \underline{S}_{\hat{J}^\star\hat{J}} && \text{for the top quadrilateral } \hat{K}^\star.\end{aligned}$$

It remains to compute the S matrices for the subdivision into four children as in Figure 3.14(c). Since only the topology and the subdivision ratio influences the S matrix, the subdivision of $\hat{K}$ into the four children $\hat{K}^a$, $\hat{K}^b$, $\hat{K}^c$ and $\hat{K}^d$ can be constructed by subdividing $\hat{K}$ horizontally into two children and subdividing both children vertically into two children. This is reflected by concatenating the S matrices of the two subdivision processes above:

$$\begin{aligned}\underline{S}_{\hat{K}^d\hat{K}} &= \big(\underline{S}_{\hat{J}'\hat{J}} \otimes \mathbb{I}\big) \cdot \big(\mathbb{I} \otimes \underline{S}_{\hat{J}^\star\hat{J}}\big), & \underline{S}_{\hat{K}^c\hat{K}} &= \big(\underline{S}_{\hat{J}^\star\hat{J}} \otimes \mathbb{I}\big) \cdot \big(\mathbb{I} \otimes \underline{S}_{\hat{J}^\star\hat{J}}\big), \\ \underline{S}_{\hat{K}^a\hat{K}} &= \big(\underline{S}_{\hat{J}'\hat{J}} \otimes \mathbb{I}\big) \cdot \big(\mathbb{I} \otimes \underline{S}_{\hat{J}'\hat{J}}\big), & \underline{S}_{\hat{K}^b\hat{K}} &= \big(\underline{S}_{\hat{J}^\star\hat{J}} \otimes \mathbb{I}\big) \cdot \big(\mathbb{I} \otimes \underline{S}_{\hat{J}'\hat{J}}\big).\end{aligned}$$

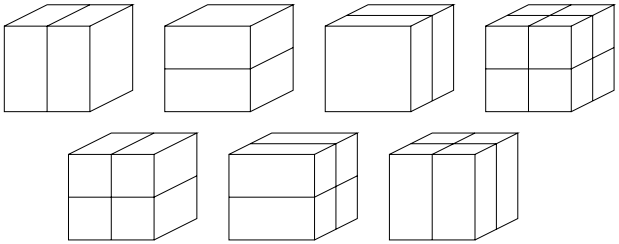


Figure 3.15: Subdivision variants for a hexahedron in three dimensions.

Remark 3.41 *It is possible to simplify the above results for the S matrices of the subdivision into four children. For instance,*

$$\underline{S}_{\hat{K}^c\hat{K}} = \underline{S}_{\hat{\jmath}\star\hat{\jmath}} \otimes \underline{S}_{\hat{\jmath}\star\hat{\jmath}},$$

but the original formulation has a much simpler implementation: A matrix tensor product where one factor is an identity matrix is easier to implement than a general matrix tensor product and the computational efficiency is not spoiled. This is so, especially, if only the application of the whole product is sought.

S Matrix in Three Dimensions

With the same idea which was used to derive the two dimensional S matrices, the S matrices in three and higher dimensions can be derived from the one dimensional S matrices with matrix tensor products.

In three dimensions, the S matrices for the subdivision shown in Figure 3.15 into two (three variants), four (three variants) or eight children take the form

$$\underline{S}_{\hat{K}'\hat{K}} = \prod \left(\underline{A} \otimes \underline{B} \otimes \underline{C}\right).$$

In each of the factors of the above product, exactly one of $\underline{A}$, $\underline{B}$ or $\underline{C}$ is a 'one dimensional' S matrix; the other two are identity matrices. A subdivision into two children results in one factor, into four children gives two factors and into eight children yields three factors in the above product.

3.3.6 Complexity Estimates

The S and T matrices make local refinements possible. This flexibility has to be paid for by a higher computational cost for the entries of the T matrices of elements in irregular parts of the mesh. We show how expensive these computations are in case of geometric vertex and edge meshes. Experiments in Section 3.4 confirm these results and approve the conjecture that the computation of the T matrices is the most expensive part in building an hp-FE space.

To compute the costs in the whole mesh, we first look a single application of an S matrix:

Lemma 3.42 (Complexity of the application of an S matrix) *The complexity t_S of the application of an S matrix to a column of the T matrix is:*

- $t_S = \mathcal{O}(p^2)$ *in* one dimension,
- $t_S = \mathcal{O}(p) \cdot t(A)$ *in* two dimensions *for an S matrix of the form $\underline{A} \otimes \mathbb{I}$, where $t(A)$ is the complexity for the application of the one dimensional S matrix $\underline{A}$,*
- $t_S = \mathcal{O}(p^2) \cdot t(A)$ *in* three dimensions *for an S matrix of the form $\underline{A} \otimes \mathbb{I} \otimes \mathbb{I}$.*

In three dimensions, the complexity of the application of an S matrix to one column of the T matrix is $\mathcal{O}(p^4)$.

Proof: In one dimension, the S matrices are nearly upper triangular (*c. f.* Example 3.36) and of size $p \times p$. Therefore, the matrix-vector product costs $t_S = \mathcal{O}(p^2)$.

In two dimensions, a T column is of size p^2. In tensor product notation, each part of size p is applied to a one dimensional S matrix (costing $t(A)$). Therefore, $t_S = \mathcal{O}(p) \cdot t(A)$. The same argument applies for the p^2 parts of size p of a T column in three dimensions. □

Proposition 3.43 (Creation of all T matrices in a geometric mesh)
Let $(\mathcal{T}^n, \boldsymbol{p}^n)$ be a three dimensional geometric mesh-degree combination with n layers generated by Algorithm 3.3 and the maximal polynomial degree p proportional to n: $p = \lceil mn \rceil$. Then, the creation of the T matrices for all elements costs $t_T = \mathcal{O}(p^7)$ and $t_T = \mathcal{O}(p^8)$, in a geometric vertex and geometric edge mesh, respectively.

Proof: The number of layers n is equal to the maximal polynomial degree p in the mesh-degree combination $(\mathcal{T}^n, \boldsymbol{p}^n)$.

In elements without hanging nodes, the T matrix is computed in $\mathcal{O}(p^3)$. Therefore, consider elements with hanging nodes. In an element with hanging nodes, there are $\mathcal{O}(p^2)$ T columns modified via an S matrix with costs of $\mathcal{O}(p^4)$. In a geometric vertex mesh, there are p layers, therefore, $t_T = \mathcal{O}(p^{2+4+1})$. In a geometric edge mesh, there are p layers and levels, therefore, $t_T = \mathcal{O}(p^{2+4+2})$. The number of elements per layer is bounded by L_0 independently of p (*c. f.* Proposition 2.28). □

Remark 3.44 *We do not consider geometric boundary layer meshes as there are no hanging nodes in such a mesh (c. f. Figures 3.1 and 3.24).*

3.4 Numerical Experiments

In the first numerical experiments, we compare the measured run-time costs with the complexity estimates given in Section 3.3.6. Then, some convergence results for the reaction diffusion equation in two and three dimensions are presented.

3.4.1 Run-Time Cost Analysis

Figures 3.16 and 3.17 show plots of the run-time costs of the hp-FE space generation algorithm and the stiffness matrix generation. The time measurements are plotted versus the number of degrees of freedom (short 'ndof') and the number of layers (up to 15) for three different meshes in the unit cube $D = (0,1)^3$: a geometric vertex mesh (*c. f.* Figure 3.3), a geometric edge mesh (*c. f.* Figure 3.2 on the right) and a geometric boundary layer mesh (*c. f.* Figure 3.1 on the right). The former two are irregular meshes while the latter is a regular mesh (no hanging nodes).

The plots in Figure 3.16 show the run-time costs of the two most expensive parts of the hp-FE space generation algorithm (element creation including T matrix generation and the topological searches, *c. f.* Algorithm 6.3) and its total time. There are more tasks in the hp-FE space generation algorithm than the two analysed here but their run-time cost is barely noticeable compared to the computation of the T matrices.

More precise estimates on the asymptotic behaviour of the run-time as shown in Figure 3.16) can be found in Table 3.1: It shows an estimate for the order of complexity of the most expensive part of the hp-FE space generation algorithm,

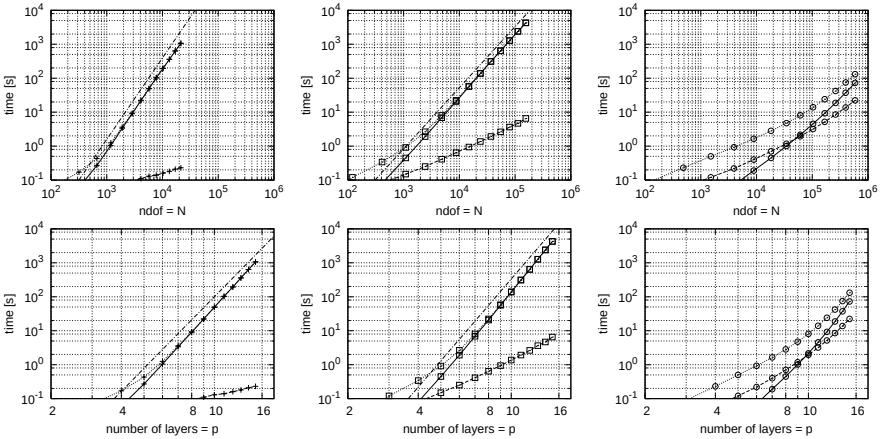


Figure 3.16: Run-time cost analysis of the hp-FE space generation algorithm plotted versus number of degrees of freedom (top) and number of layers (bottom). Plotted are the total space build time (dotted line), topological searches (dashed line) and creation of elements and T matrices (solid line) for three different meshes: vertex mesh (+, left plots), edge mesh (□, middle plots) and boundary layer mesh (◯, right plots).
The dash-dotted lines show $\mathcal{O}(N^{2.4})$ and $\mathcal{O}(p^7)$ for the vertex mesh (+, left plots) and $\mathcal{O}(N^{1.8})$ and $\mathcal{O}(p^8)$ for the edge mesh (□, middle plots) respectively for the number of degrees of freedom N and the number of layers n (which is equal to the maximal polynomial degree p).

the creation of elements and T matrices (the solid line in Figure 3.16). To estimate the order, we use the Ansatz $t(x) = \alpha \cdot x^{\beta}$ where $t(x)$ is the run-time of the algorithm and x is either the number of layers (= highest polynomial degree) p or the number of degrees of freedom N. Then,

$$\beta = \frac{\log t(x_2) - \log t(x_1)}{\log x_2 - \log x_1}.$$

The values of β of the boundary layer mesh are not very meaningful as the algorithm is still in its pre-asymptotic range for the values of N and p shown.

The results in Table 3.1 comply with the theoretical estimates of the complexity of the creation of all T columns in Proposition 3.43, namely $\mathcal{O}(p^7)$ and $\mathcal{O}(p^8)$ for the geometric vertex and edge mesh respectively.

β	N	t [s]	p	β
2.442	5706	49.090	10	7.734
2.287	7716	102.600	11	7.354
2.340	10207	194.570	12	7.641
2.327	13256	358.680	13	7.715
2.215	16947	635.370	14	7.446
	21371	1062.089	15	

β	N	t [s]	p	β
1.873	23841	137.110	10	8.539
1.818	36813	309.430	11	8.359
1.847	54916	640.410	12	8.561
1.825	79571	1270.779	13	8.521
1.792	112455	2389.701	14	8.422
	155519	4272.718	15	

β	N	t [s]	p	β
1.426	61080	2.159	10	7.839
1.469	103140	4.559	11	8.141
1.575	167041	9.260	12	8.793
1.663	261158	18.719	13	9.349
1.708	396135	37.429	14	9.666
	585278	72.920	15	

Table 3.1: Estimates for the order of complexity of the algorithm for the creation of elements and T matrices with respect to the number of degrees of freedom N in the left hand part of each table and the number of layers p in the right hand part. The top left and top right tables show the numbers of the vertex and edge mesh respectively. The bottom table shows the numbers of the boundary layer mesh. The time t is measured in seconds (information on the CPU used and how the time measurements are conducted are given in Section7.3).

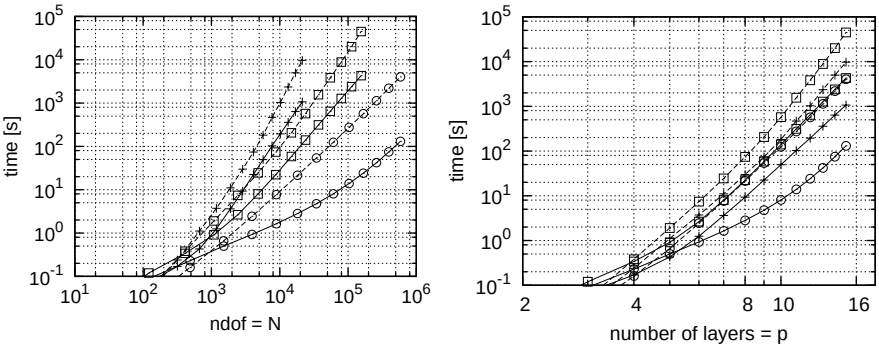


Figure 3.17: Run-time cost analysis of the hp-FE space generation algorithm and stiffness matrix computation plotted versus number of degrees of freedom (left) and number of layers (right). Plotted are the total space build time (solid line) and the time for the stiffness matrix generation (dashed line) for three different meshes: vertex mesh (+), edge mesh (□) and boundary layer mesh (◯).

To give an impression how the run-time costs for the hp-FE space generation relate to other typical tasks in a FE simulation, Figure 3.17 shows a comparison of the run-time costs of the space generation with the integration and assembling of the stiffness matrix

$$\int_D \nabla u \cdot \nabla v \, d\boldsymbol{x}.$$

The stiffness matrix evaluation takes about ten times longer on an irregular mesh—on a regular mesh, the difference is even more pronounced.

3.4.2 Reaction Diffusion Equation

In this section, some numerical results of the reaction diffusion equation (introduced in Section 1.3.1) in two and three dimension are shown.

The convergence graphs are shown in the relative energy error versus a power of the number of degrees of freedom (short 'ndof'). If $a(.\,,.)$ is symmetric, the energy error satisfies [106]:

$$\begin{aligned}
\|u_{\text{exact}} - u_N\|_E^2 &= a(u_{\text{exact}} - u_N, u_{\text{exact}} - u_N) \\
&= a(u_{\text{exact}} - u_N, u_{\text{exact}}) - \underbrace{a(u_{\text{exact}} - u_N, u_N)}_{\substack{=0 \\ \text{Galerkin orthogonality (1.6)}}} \\
&= a(u_{\text{exact}}, u_{\text{exact}}) - \underbrace{a(u_N, u_{\text{exact}})}_{=l(u_N)=a(u_N,u_N)} \\
&= \|u_{\text{exact}}\|_E^2 - \|u_N\|_E^2,
\end{aligned}$$

where $u_N = \boldsymbol{\Phi}^\top \boldsymbol{u}$ is the Finite Element solution and $\|u_{\text{exact}}\|_E^2$ is known exactly (by symbolic integration performed by Mathematica or an overkill computation using the same program). $\|u_N\|_E^2 = a(u_N, u_N)$ where $a(.\,,.)$ is the bilinear form associated to the problem is computed by $a(u_N, u_N) = \boldsymbol{u}^\top \underline{A} \boldsymbol{u}$. In the plots, the relative energy error

$$\frac{\|u_{\text{exact}}\|_E^2 - \|u_N\|_E^2}{\|u_{\text{exact}}\|_E^2} \tag{3.45}$$

is shown.

Two Dimensions

The problem solved in the following to sections is always

$$\begin{aligned} -a\Delta u + cu &= f && \text{in } D \subset \mathbb{R}^2, \\ u &= 0 && \text{on } \Gamma_D, \\ \frac{\partial u}{\partial \boldsymbol{n}} &= g && \text{on } \Gamma_N \end{aligned}$$

on various domains D and with various sets of data a, c, f and g.

Problem with a Boundary Layer The domain D is the unit square $(0,1)^2$, *c. f.* Figure 3.18 and $\Gamma_N = \emptyset$, $\Gamma_D = \partial D$. The right hand side f and the reaction coefficient c are chosen to be 1. The diffusion coefficient a is chosen much smaller than 1. This results in a boundary layer with a thickness of $\mathcal{O}(\sqrt{a})$ near the Dirichlet boundary Γ_D [83, 98]:

$$\begin{aligned} -a\Delta u + u &= 1 && \text{in } D = (0,1)^2 \subset \mathbb{R}^2, \\ u &= 0 && \text{on } \partial D. \end{aligned}$$

Figures 3.19 and 3.20 show the convergence histories of the relative energy error (3.45) with $a = 10^{-2}$ and $a = 10^{-12}$ respectively. In the latter case, the energy of the exact solution is taken to be 1 as the boundary layer is thin: the error introduced by this is estimated to be of order $\sqrt{10^{-12}} = 10^{-6}$ which is much smaller than the achieved accuracy of order 10^{-4}.

Remark 3.46 *Both convergence plots in Figures 3.19 and 3.20 give the convergence in* $\log-\sqrt[4]{\cdot}$ *scale although Proposition 2.33 predicts that the error was of the order* $\mathcal{O}(\exp(-bN^{1/3}))$. *However, this is no longer true for boundary layer meshes, where the number of degrees of freedom* N *behaves like* $\mathcal{O}(p^4)$ *because of the layers and levels in the edge and corner neighbourhood (c. f. the situation in three dimensions described in Section 2.3).*

Singular Behaviour of the Solution The domain D is the L shaped domain $(-1,1)^2 \setminus (0,1) \times (-1,0)$, *c. f.* Figure 3.21 for the partitioning of ∂D into Γ_D and

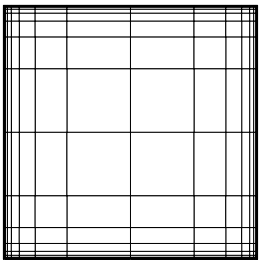


Figure 3.18: Geometric boundary layer mesh in the unit square generated using Algorithm 3.2 with all edges marked as singular. The (anisotropic) degree p_K is linearly distributed with $m = 1$.

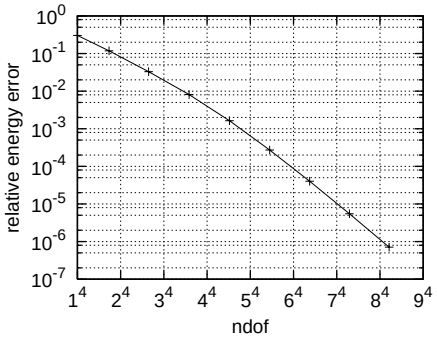


Figure 3.19: Convergence of the relative energy error versus number of degrees of freedom in log–$\sqrt[4]{\cdot}$ scale for the boundary layer problem with $a = 10^{-2}$: the straight line shows exponential convergence.

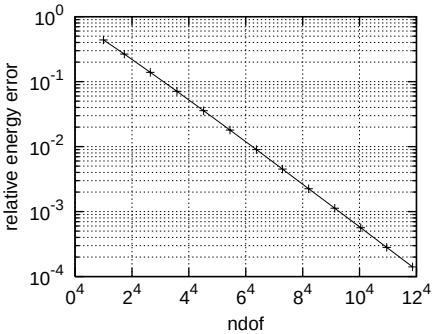


Figure 3.20: Convergence of the relative energy error versus number of degrees of freedom in log–$\sqrt[4]{\cdot}$ scale for the boundary layer problem with $a = 10^{-12}$: the straight line shows exponential convergence.

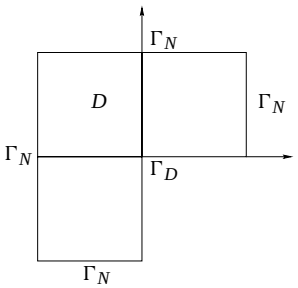


Figure 3.21: Boundary conditions for the L shaped domain.

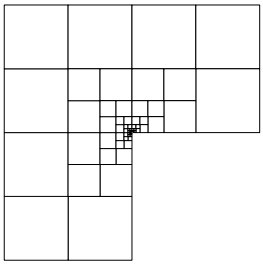


Figure 3.22: Geometric vertex mesh in the L shaped domain generated using Algorithm 3.2 with the reentrant corner marked as singular. The (isotropic) degree p_K is linearly distributed with $m = 1$.

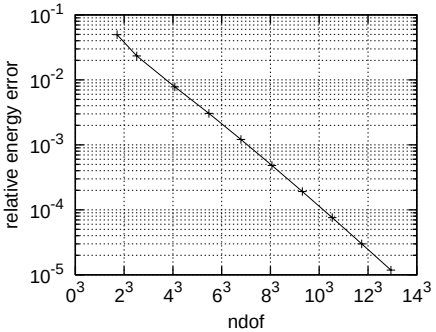


Figure 3.23: Convergence of the relative energy error versus number of degrees of freedom in log–$\sqrt[3]{\cdot}$ scale for the two dimensional problem with a singularity: the straight line shows exponential convergence.

Γ_N. The right hand side f and the reaction coefficient c are chosen to be 0. With the following Neumann boundary conditions

$$\frac{\partial u}{\partial \boldsymbol{n}} = \frac{2}{3r^{4/3}} \left[\begin{pmatrix} x \\ y \end{pmatrix} \sin(2/3\varphi) + \begin{pmatrix} -y \\ x \end{pmatrix} \cos(2/3\varphi) \right]. \qquad \text{on } \Gamma_N,$$

the problem $-\Delta u = 0$ in the L shaped domain D has the exact solution $u = r^{2/3} \sin(2/3\varphi)$, where (r, φ) are the polar coordinates with respect to the origin. The mesh and the convergence history are shown in Figures 3.22 and 3.23 respectively.

Three Dimensions

As in two dimensions, the following problem is solved:

$$\begin{aligned} -a\Delta u + cu &= f && \text{in } D \subset \mathbb{R}^2, \\ u &= 0 && \text{on } \Gamma_D, \\ \frac{\partial u}{\partial \boldsymbol{n}} &= g && \text{on } \Gamma_N \end{aligned}$$

on different domains D and with different data a, c, f and g.

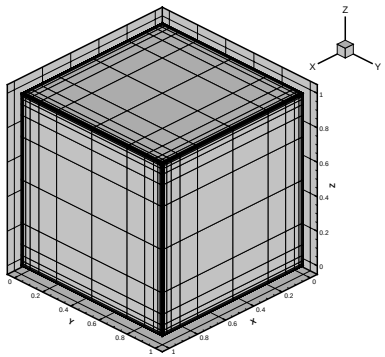


Figure 3.24: Geometric boundary layer mesh in the unit cube generated using Algorithm 3.2 with the sides marked as singular. The (anisotropic) degree p_K is linearly distributed with $m = 1$.

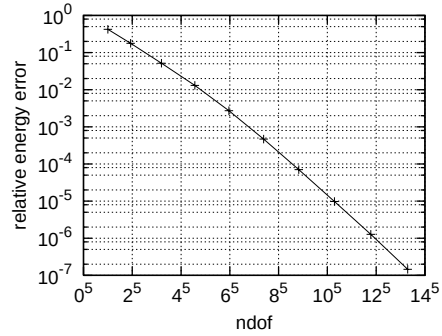


Figure 3.25: Convergence of the relative energy error versus number of degrees of freedom in log–$\sqrt[5]{\cdot}$ scale for the three dimensional boundary layer problem: the straight lines shows exponential convergence.

Problem with a Boundary Layer The domain D is the unit cube $(0,1)^3$, *c.f.* Figure 3.24 and $\Gamma_N = \emptyset$, $\Gamma_D = \partial D$. The right hand side f and the reaction coefficient c are chosen to be 1. The diffusion coefficient a is chosen much smaller than 1. This results in a boundary layer with a thickness of $\mathcal{O}(\sqrt{a})$ near the Dirichlet boundary Γ_D [83, 98]:

$$\begin{aligned} -a\Delta u + u &= 1 && \text{in } D = (0,1)^3 \subset \mathbb{R}^3, \\ u &= 0 && \text{on } \partial D. \end{aligned}$$

Figure 3.25 shows the convergence with $a = 10^{-2}$ while Figure 3.24 shows the mesh.

Singular Behaviour of the Solution (Vertex Singularity) The problem in three dimensions is similar to the problem with the singularity in two dimensions. The domain D is the Fichera corner $(-1,1)^3 \setminus (-1,0)^3$. The reentrant corner (located at the origin) is marked as singular since it is known that the exact solution

$$u = \sqrt{r}\sin(\varphi)\sin(\theta)$$

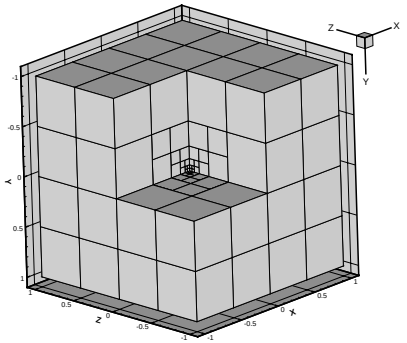


Figure 3.26: Geometric vertex mesh in the Fichera corner generated using Algorithm 3.2 with the reentrant corner marked as singular. The (isotropic) degree p_K is linearly distributed with $m = 1$.

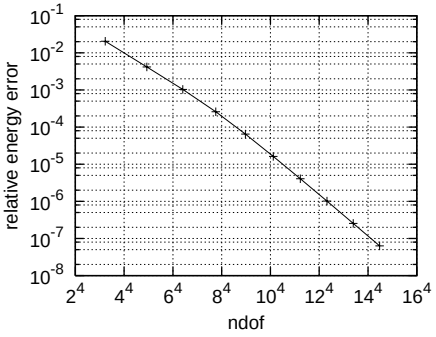


Figure 3.27: Convergence of the relative energy error versus number of degrees of freedom in log–$\sqrt[4]{\cdot}$ scale for the three dimensional problem with a vertex singularity: the straight line shows exponential convergence.

has a singularity there. The coordinates (r, φ, θ) are the usual spherical coordinates with respect to the origin.

The problem solved is

$$
\begin{aligned}
-\Delta u + u &= \sqrt{r}\sin(\varphi)\sin(\theta)\left(1 + \frac{5}{4x^2}\right) && \text{in } D = (-1,1)^3 \setminus (-1,0)^3,\\
\frac{\partial u}{\partial \boldsymbol{n}} &= \begin{pmatrix} \frac{\sin(\theta)\sin(\varphi)}{2\sqrt{r}} \\ \frac{\cos(\theta)\sin(\varphi)}{\sqrt{r}} \\ \frac{\cos(\varphi)}{\sqrt{r}} \end{pmatrix} && \text{on } \Gamma_N,\\
u &= 0 && \text{on } \Gamma_D,
\end{aligned}
$$

where only the quadrilateral with the corners $(0,0,-1)$, $(-1,0,-1)$, $(-1,0,0)$ and $(0,0,0)$ has the homogeneous Dirichlet boundary condition. The mesh and the convergence history are shown in Figures 3.26 and 3.27 respectively.

Remark 3.47 *Figure 3.27 gives the convergence history in* log–$\sqrt[4]{\cdot}$ *scale although Proposition 2.69 states that the error should be of the order* $\mathcal{O}(\exp(-bN^{1/5}))$. *However, there are no edge singularities in this problem. Therefore, there are no*

layers and levels in the corner and edge neighbourhood, i. e. the number of degrees of freedom is of order $\mathcal{O}(p^4)$.

Part II

Applications

4 Maxwell's Equations

Maxwell's equations are used for modelling electro-magnetic wave phenomena. The main physical quantities in Maxwell's equations are the electric and magnetic fields. There are problems where they can have unbounded singularities (*c. f.* the numerical experiments in Section 4.3). For this reason, the numerical treatment of Maxwell's equations was classically done using Nédélec's edge elements [87, 88]. However, implementing such elements is not trivial. In addition, proving p- or hp-convergence for Nédélec's edge element approximations is still an open problem in three dimensions (the two dimensional case is studied in [5]). Recently, Costabel and Dauge have introduced the weighted regularisation to overcome both problems [29]. New results show that hp-FEM with weighted regularisation are able to resolve Maxwell problems at an exponential rate of convergence [31, 32]—we give numerical evidence for these results.

This chapter briefly reviews the weighted regularisation for Maxwell's equations in its second section. The first section reviews the variational formulation already presented in Section 1.3.3. The final section shows numerical results in various domains in two and three dimensions. There, numerical evidence is given for the conjectured exponential convergence of the Eigenvalues in terms of degrees of freedom.

4.1 Time Harmonic Maxwell's Equations

The time harmonic Maxwell's Equations are given in Section 1.3.3. Formally, the *electric source problem* reads (1.24)

$$\begin{aligned} \mathbf{curl}(\mu^{-1}\,\mathbf{curl}\,\boldsymbol{E}) - \omega^2 \underbrace{\left(\varepsilon + \frac{\sigma}{i\omega}\right)}_{=:\tilde{\varepsilon}} \boldsymbol{E} &= -i\,w\,\boldsymbol{J}, \\ \operatorname{div} \varepsilon \boldsymbol{E} &= 0. \end{aligned} \tag{4.1}$$

Eigensolutions of the electric problem (4.1) are found for $\boldsymbol{J} \equiv 0$ and $\sigma = 0$:

$$\mathbf{curl}(\mu^{-1}\,\mathbf{curl}\,\boldsymbol{E}) - \omega^2 \varepsilon \boldsymbol{E} = 0. \tag{4.2}$$

The *variational formulation* for the electric source problem reads:
Find $\boldsymbol{E} \in H_0(\mathbf{curl};\, D)$ such that

$$\int_D \left(\mu^{-1}\,\mathbf{curl}\,\boldsymbol{E} \cdot \mathbf{curl}\,\boldsymbol{v} - \omega^2 \tilde{\varepsilon}\boldsymbol{E} \cdot \boldsymbol{v}\right) d\boldsymbol{x} = \int_D \boldsymbol{f} \cdot \boldsymbol{v}\, d\boldsymbol{x} \quad \forall \boldsymbol{v} \in H_0(\mathbf{curl};\, D), \tag{4.3}$$

where

$$H(\mathbf{curl};\, D) := \{\boldsymbol{u} \in L^2(D)^d : \mathbf{curl}\,\boldsymbol{u} \in L^2(D)^d\},$$

with the *perfect conductor boundary conditions* (1.28):

$$H_0(\mathbf{curl};\, D) := \{\boldsymbol{u} \in H(\mathbf{curl};\, D) : \boldsymbol{u} \times \boldsymbol{n} = 0 \text{ on } \partial D\}. \tag{4.4}$$

4.2 Weighted Regularisation

A well-known strategy for Finite Element computations of the Maxwell Eigenfrequencies is the use of 'spurious-free' elements whose classical representatives are the two families of *Nédélec's edge elements* [87, 88]. There are also good reasons why one may prefer a discretisation of the Maxwell problem which uses standard and widely used elements like nodal elements. In this case, the compatibility conditions between neighbouring elements are point-wise and scalar as opposed to

tangential and vector valued in the Nédélec case. The idea [67, 79] is to penalise the divergence by adding (assume ε constant)

$$\int_D \operatorname{div} \boldsymbol{u} \operatorname{div} \boldsymbol{v} \, d\boldsymbol{x}$$

to the variational form and introducing the variational space

$$X_n := \{\boldsymbol{u} \in H_0(\mathbf{curl}; D) : \operatorname{div} \boldsymbol{u} \in L^2(D)\}. \tag{4.5}$$

This method has drawbacks in domains with reentrant corners [27, 28, 30]: *it does not capture the correct solution.* The new idea developed in [29] is to introduce intermediate spaces $X_n[Y]$ between (4.4) and (4.5) coupled with the corresponding modification of the bilinear form in (4.3), such that the space $H_n := H^1(D)^d$ is dense in $X_n[Y]$, the associated operator is elliptic and the solution in $X_n[Y]$ of the new problem coincides with that of (4.1) (the proof is given in [29], Section 2.1).

The new bilinear form with *weighted regularisation* is

$$a_Y(\boldsymbol{u}, \boldsymbol{v}) := \int_D \left(\mu^{-1} \mathbf{curl}\, \boldsymbol{u} \cdot \mathbf{curl}\, \boldsymbol{v} - \omega^2 \tilde{\varepsilon} \boldsymbol{u} \cdot \boldsymbol{v}\right) d\boldsymbol{x} + \langle \operatorname{div} \boldsymbol{u}, \operatorname{div} \boldsymbol{v} \rangle_Y \tag{4.6}$$

where Y is a space such that H_n is dense in

$$\begin{aligned} X_n[Y] &:= \{\boldsymbol{u} \in H_0(\mathbf{curl}; D) : \operatorname{div} \boldsymbol{u} \in Y\}, \\ \|u\|^2_{X_n[Y]} &:= \|\mathbf{curl}\, \boldsymbol{u}\|^2_{L^2(D)^d} + \langle \operatorname{div} \boldsymbol{u}, \operatorname{div} \boldsymbol{u} \rangle_Y + \|u\|^2_{L^2(D)^d} \end{aligned}$$

and $L^2(D) \subset Y \subset H^{-1}(D)$. Therefore, it suffices to concentrate on weighted L^2 spaces:

$$\{\varphi \in L^2_{\mathrm{loc}}(D) : w\varphi \in L^2(D)\}$$

with a weight $w \in \mathcal{C}^\infty(D)$. $a_Y(\boldsymbol{u}, \boldsymbol{v})$ is a continuous, coercive bilinear form for $\omega = 0$. Therefore, the source problem (4.3) with the bilinear form $a_Y(\boldsymbol{u}, \boldsymbol{v})$ fits into the general framework of Section 1.1.

A simple choice for the weight w in three dimensions is

$$w = \operatorname{dist}\Big(\boldsymbol{x}, \bigcup_{\boldsymbol{c} \in \mathcal{C}} \boldsymbol{c} \cup \bigcup_{e \in \mathcal{E}} \overline{e}\Big). \tag{4.7}$$

Other formulations for the weight w are possible, as long as they are norm-equivalent to (4.7) (*c. f.* Section 4.2.1 below). Most importantly, it is possible to limit the weight to 1 far away from singularities.

Plugging the weight into (4.6) yields

$$\langle \operatorname{div} \boldsymbol{u}, \operatorname{div} \boldsymbol{v} \rangle_Y = \int_D w^2 \operatorname{div} \boldsymbol{u} \cdot \operatorname{div} \boldsymbol{v} \, d\boldsymbol{x} = \int_D \operatorname{dist}\Big(\boldsymbol{x}, \bigcup_{\boldsymbol{c} \in \mathcal{C}} \boldsymbol{c} \cup \bigcup_{e \in \mathcal{E}} \overline{e}\Big)^2 \operatorname{div} \boldsymbol{u} \cdot \operatorname{div} \boldsymbol{v} \, d\boldsymbol{x}$$

for the simple weight (4.7).

The Maxwell Eigenproblem is solved combing (4.2), (4.3) and (4.6):
Find frequencies ω and functions $0 \neq \boldsymbol{E} \in X_n[Y]$ such that

$$\int_D \mu^{-1} \operatorname{\mathbf{curl}} \boldsymbol{E} \cdot \operatorname{\mathbf{curl}} \boldsymbol{v} \, d\boldsymbol{x} + \langle \operatorname{div} \boldsymbol{E}, \operatorname{div} \boldsymbol{v} \rangle_Y = \omega^2 \int_D \varepsilon \boldsymbol{E} \cdot \boldsymbol{v} \, d\boldsymbol{x} \quad \forall \boldsymbol{v} \in X_n[Y]. \tag{4.8}$$

(4.8) is a real Eigenproblem if the conductivity $\sigma = 0$, *i. e.* there are only insulators in the domain D. If $X_n[Y] \subset L^2(D)^d = H$ is a compact embedding, we can conclude that (4.8) fits into the setting of Section 1.3.4. However, the compactness of $X_n[Y]$ in $L^2(D)^d$ is not straight forward—it is discussed in Section 4.2.1 below.

Remark 4.9 (4.8) *can be solved using standard nodal FEM in each component of the solution* $\boldsymbol{E}$ *which discretise* H_n. *If the boundary consists only of axi-parallel planes, the perfect conductor boundary conditions* (1.28) *can be applied as carefully chosen Dirichlet boundary conditions to each of the three components of* $H_n = H^1(D)^3$. *The same applies to a two dimensional polygon with axi-parallel edges.*

According to [29], $H_n \overset{\text{dense}}{\subset} X_n[Y]$. Therefore, using standard Finite Element functions $u_N \in V_N = \mathcal{S}^{\boldsymbol{p},1}_{\Gamma_D}(\mathcal{T}, D)^3$ can be used to approximate functions in $X_n[Y]$. Suitable refinements of V_N generate convergent series of Galerkin approximations. Assuming $X_n[Y] \overset{\text{compact}}{\subset} L^2(D)^d$, this implies that also Eigenvalues and Eigenfunctions of (4.8) can be approximated and the FE approximations converge to the true solution.

On the other hand, convergence of Nédélec's edge elements using p- or hp-extensions is not proven in three dimensions. However, there exists numerical evidence for such a convergence [4, 33, 44].

4.2.1 Selection of Weights in the Weighted Regularisation

In this section, some more details on the weights are presented, following [29].

First, in each of the subdomains introduced in Sections 2.2.1 (two dimensions) and 2.3.1 (three dimensions) except $\mathcal{V}^0$, a different weight w_i should be applied. In addition, in three dimensions, using the distance function ρ_e (blowing up at the corners), the edges are also isolated from each other. Therefore, the product of all individual weight functions $\prod w_i$ is equivalent to the weight w_i in each of the subdomains.

Two Dimensional Weights

The domains $\mathcal{V}_{\boldsymbol{a}}$ (vicinity of a corner $\boldsymbol{a}$) and $\mathcal{V}^0$ (regular part) and the distance function $r_{\boldsymbol{a}}(\boldsymbol{x}) = |\boldsymbol{x} - \boldsymbol{a}|$ referenced in this section are the same as in Section 2.2.1.

The regularity in $\mathcal{V}_{\boldsymbol{a}}$ implies [29] that the weight should be chosen as

$$r_{\boldsymbol{a}}^{\gamma_{\boldsymbol{a}}}. \tag{4.10}$$

The exponent $\gamma_{\boldsymbol{a}}$ in (4.10) is

$$0 \leq 1 - \frac{\pi}{\omega_{\boldsymbol{a}}} < \gamma_{\boldsymbol{a}} \leq 1. \tag{4.11}$$

Therefore, a global weight is

$$w = \prod_{\boldsymbol{a} \in \mathcal{C}} r_{\boldsymbol{a}}^{\gamma_{\boldsymbol{a}}}. \tag{4.12}$$

The different $\gamma_{\cdot}$ can be simplified to $\gamma_{\max}$:

$$w = \Big(\prod_{\boldsymbol{a} \in \mathcal{C}} r_{\boldsymbol{a}} \Big)^{\gamma_{\max}}. \tag{4.13}$$

The following, simple weight w is equivalent to (4.12):

$$w = \operatorname{dist}\Big(\boldsymbol{x}, \bigcup_{\boldsymbol{a} \in \mathcal{C}} \boldsymbol{a} \Big)^{\gamma_{\max}}. \tag{4.14}$$

Remark 4.15 (Compactness of $X_n[Y] \subset L^2(D)^d$) *In the limiting case $\gamma_{\boldsymbol{a}} = 1$ in* (4.11)*, $w^{\gamma_{\boldsymbol{a}}}$ in* (4.10) *is polynomial. This property is desired as it makes the evaluation of $w^{2\gamma_{\boldsymbol{a}}}$ in $a_Y(\boldsymbol{u}, \boldsymbol{v})$ less expensive and the numerical integration more accurate. However, the compactness of the embedding $X_n[Y]$ into $L^2(D)^d$ is lost for $\gamma_{\boldsymbol{a}} = 1$ [29]. Therefore, convergence of discrete Eigensolution to the true solutions can only be asserted for $\gamma_{\boldsymbol{a}} < 1$.*

Three Dimensional Weights

The domains $\mathcal{V}_{\boldsymbol{c}}^0$, $\mathcal{V}_e(\boldsymbol{c})$, $\mathcal{V}_e^0$, $\mathcal{V}^0$ and distance functions $r_{\boldsymbol{c}}$, r_e, ρ_e referenced in this section are the same as in Section 2.3.1.

The regularity in the different domains $\mathcal{V}_{\boldsymbol{c}}^0$, $\mathcal{V}_e^0$ and $\mathcal{V}_e(\boldsymbol{c})$ implies [29] that the weights should be chosen as

$$r_{\boldsymbol{c}}^{\gamma_{\boldsymbol{c}}} \text{ in } \mathcal{V}_{\boldsymbol{c}}^0, \qquad r_e^{\gamma_e} \text{ in } \mathcal{V}_e^0, \qquad r_{\boldsymbol{c}}^{\gamma_{\boldsymbol{c}}} \cdot \rho_e^{\gamma_e} \text{ in } \mathcal{V}_e(\boldsymbol{c}). \tag{4.16}$$

The exponents γ_e and $\gamma_{\boldsymbol{c}}$ in (4.16) are

$$0 \leq 1 - \min_{\boldsymbol{x}\in\overline{e}} \frac{\pi}{\omega_e(\boldsymbol{x})} < \gamma_e \leq 1, \qquad 0 \leq 1/2 - \lambda_{\boldsymbol{c},1}^{\mathrm{Dir}} < \gamma_{\boldsymbol{c}} \leq 1,$$

where

$$\lambda_{\boldsymbol{c},\pm k}^{\mathrm{Dir}} = -1/2 \pm \sqrt{\mu_k^{\mathrm{Dir}} + 1/4}$$

and μ_k^{Dir} the Eigenvalues of the Laplace-Beltrami operator on $G_{\boldsymbol{c}}$ (*c.f.* Section 2.3.1) with Dirichlet boundary conditions.

To sum up, a global weight is

$$w = \Big(\prod_{\boldsymbol{c}\in\mathcal{C}} r_{\boldsymbol{c}}^{\gamma_{\boldsymbol{c}}}\Big) \cdot \Big(\prod_{e\in\mathcal{E}} \rho_e^{\gamma_e}\Big). \tag{4.17}$$

(4.17) is equivalent to (4.7). The different $\gamma_{\cdot}$ can be simplified to $\gamma_{\max}$:

$$w = \Big(\prod_{\boldsymbol{c}\in\mathcal{C}} r_{\boldsymbol{c}} \cdot \prod_{e\in\mathcal{E}} \rho_e\Big)^{\gamma_{\max}}. \tag{4.18}$$

Remark 4.15 also holds in the three dimensional case.

4.2.2 Computation of Weights

In the computation of the div-div element matrix, the weight $w^{2\gamma}(\boldsymbol{x})$ is computed together with the determinant of the Jacobian of the element map $\det F_K'(\boldsymbol{x})$ at every quadrature point. This is used during the numerical integration with the quadrature rule. The evaluation of the weight $w(\boldsymbol{x})$ is done by an external routine

which is given as a template argument to the DivDiv class at compile time.[1] This routine needs as input the cell K and the reference coordinates $\boldsymbol{\xi} \in \hat{K}$ and returns the value of w at this specific point.

Example 4.19 *The simplest weight routine* TrivialWeight *always returns 1. The compiler is able to completely eliminate the call to the weight function in this case. This is for computations without weight. There are two other implementations of a weight function:* ShortestDist *implementing* (4.7) *and* ProductOfAll *implementing* (4.18) *for* $\gamma_{\max} = 1$. *In the case of* $\gamma_{\max} < 1$, *a post-processing routine nested around the weight raises* w *to the given power.*

Both ShortestDist *and* ProductOfAll *know about the corners and edges with singularities and compute the distances to these. In the case of* ShortestDist, *the shortest distance is returned and in the case of* ProductOfAll, *the product of all distances is returned.*

In this way, the spaces $X_n[Y]$ can be implemented easily with good control of the used weight. It can be concluded from [29] that $X_n[Y]$ with the weight w given in (4.7) and (4.18) and the sketched implementation satisfy all assumptions required to capture the correct solutions of Maxwell's equations in the formulation (4.8).

4.3 Numerical Results

The numerical results for the benchmark problems [36] of (4.8) are obtained using using Concepts (*c. f.* Part III) in two and three dimensions. The actual equation solved here has an additional parameter s in front of the $\langle .\,, .\rangle_Y$-form:
Find frequencies ω and functions $0 \neq \boldsymbol{E} \in X_n[Y]$ such that

$$\int_D \mu^{-1}\, \mathbf{curl}\, \boldsymbol{E} \cdot \mathbf{curl}\, \boldsymbol{v}\, d\boldsymbol{x} + s\langle \operatorname{div} \boldsymbol{E}, \operatorname{div} \boldsymbol{v}\rangle_Y = \omega^2 \int_D \varepsilon\, \boldsymbol{E} \cdot \boldsymbol{v}\, d\boldsymbol{x} \quad \forall \boldsymbol{v} \in X_n[Y]. \tag{4.20}$$

This scaling parameter $s > 0$ can be chosen arbitrarily—it helps finding spurious Eigenvalues. Physical Eigenfunctions $\boldsymbol{E}$ have $\operatorname{div} \boldsymbol{E} \equiv 0$ and the $\langle .\,, .\rangle_Y$-form disappears from (4.20) whereas spurious (non-physical) Eigenfunctions have

[1] The template technique lessens the runtime flexibility: The user cannot easily be prompted for an arbitrary weight to be used. On the other hand, the evaluation of the weight can be highly optimised by the compiler as all information is there at compile time. This is especially useful in this case as the weight function is called $\mathcal{O}(p^3)$ times in each cell K.

a non-zero divergence. The spurious Eigenvalues are scaled with this parameter s whereas the physical Eigenvalues are independent of s.

Besides figures of the domains and meshes, all numerical examples below feature plots of the convergence of the first three Eigenvalues and a plot of the Eigenvalues versus the scaling parameter s. In the Eigenvalues versus s plots, the Eigenvalues $\lambda = \omega^2$ are categorised with the following criterion using the Eigenfunction $\boldsymbol{E}$:

$$\frac{\|\,\mathbf{curl}\,\boldsymbol{E}\|_0^2}{s\|\,\mathrm{div}\,\boldsymbol{E}\|_Y^2} \begin{cases} \geq \rho & \text{physical Eigenvalue} \\ \leq \rho^{-1} & \text{spurious Eigenvalue} \\ \text{otherwise} & \text{undecided.} \end{cases} \tag{4.21}$$

The value used for ρ is 1.5—this gives good results ([37] and our own experience). The spurious Eigenvalues are lined up on straight lines through the origin in the Eigenvalue versus s plots whereas the the physical Eigenvalues are lined up on horizontal lines.

Note that we use $\gamma = 1$ in all computations and loose the compactness of $X_n[Y] \subset L^2(D)^d$ (*c.f.* Remark 4.15). Nonetheless, in our numerical experiments, we are able to achieve convergence of the discrete solutions towards the true solutions.

Below, we give numerical evidence for the exponential convergence of hp-FEM with weighted regularisation to solve Maxwell Eigenvalue problems. Very recently, Costabel, Dauge and Schwab have been able to prove the exponential convergence theoretically [31, 32].

4.3.1 Sources of Errors in Eigenvalue Computations

The numerical results presented below are distorted by different errors.

Modelling error

We neglect this error source and take the model to be exact. The goal of the computations below is not the simulation of some physical phenomenon but studying the performance of the method and its implementation (verification of the computer program).

Discretisation error

Assuming all matrices are known exactly and all matrix Eigenvalue problems are solved exactly, the estimates in Section 1.3.4 and the Rayleigh Minimax principle apply for any value of s: $\lambda_k \leq \lambda_{k,N}$ and div $\boldsymbol{E} = 0$. By refining the hp-FE spaces, the discretisation error is reduced.

Numerical quadrature error

In all examples in this thesis, the numerical integration in the bilinear and linear forms is done using a tensor product Gauss Jacobi quadrature rule with $p+2$ integration points in reference coordinates.[2] This enables us to integrate polynomials of order $2p+3$ exactly. Assuming an affine element map F_K and therefore a constant determinant of the Jacobian in the integration in reference coordinates leaves us with $2p+3-2p=3$ orders for the integration of the weight ($2p$ is consumed by the integration of div $\boldsymbol{E} \cdot$ div $\boldsymbol{v}$).

Non-affine element maps introduce a determinant of the Jacobian into the integration in reference coordinates which is a *rational function* and therefore cannot be integrated exactly by the given rule.

Another source of quadrature errors is a non-polynomial weight w. This happens for exponents $\gamma \neq 1$ or weights including a minimum operation (*e. g.* (4.14) and (4.7) in two and three dimensions respectively). The polynomial degree of a product weight (*e. g.* (4.13) and (4.18) in two and three dimensions respectively) is (in most cases) too large to be integrated exactly by the given rule.

However, this does not harm the convergence as can seen by a Strang-type argument [103]: The integration order in large elements far from the edges and vertices in $\mathcal{E}$ and $\mathcal{C}$ respectively is increased and the small elements close to $\mathcal{E}$ and $\mathcal{C}$ are subdivided.

Eigensolver errors

For reasonably large Eigenproblems, only iterative solvers are feasible. We have chosen ARPACK [77, 78] in shift-invert mode as our Eigensolver—the linear system inside the Eigensolver is solved by Umfpack [38, 39, 40], Pardiso [91] or

[2] In case of an anisotropic polynomial degree $\boldsymbol{p}_K$, the order of the tensor product quarature rule is also anisotropic. The sum factorisation technique for efficient evaluation of the numerical integration is discussed in Section 6.4 (see also [84]).

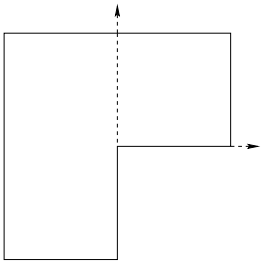


Figure 4.1: L shaped domain $(-1,1)^2 \setminus (0,1) \times (-1,0)$.

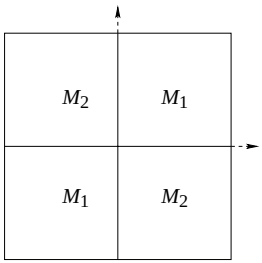


Figure 4.2: Square $(-1,1)^2$ for the transmission problem. The areas M_1 and M_2 are made of different materials.

SuperLU [45]. ARPACK has a tolerance parameter to test for convergence which we set to machine precision $2.2 \cdot 10^{-16}$. This should eliminate numerical errors in the Eigenvalue solver as much as possible.

However, ill-conditioned Eigenproblems can still create a large dispersion in the numerical results. The condition of the Eigenproblem depends on the distance of the Eigenvalue to the rest of the spectrum of the operator [22]. When a spurious Eigenmode comes close to a physical Eigenmode, the condition number is large. This is the reason for non-monotonicity in the convergence histories of the Eigenvalues. Large condition numbers increase the influence of distortions of the operator which are introduced by inexact numerical integration.

4.3.2 Two Dimensions

The L shaped domain (Figure 4.1) and a square for transmission problems (two different materials M_1 and M_2 in a checkerboard pattern, Figure 4.2) are treated. In both problems, singularities arise. The different materials M_1 and M_2 in the transmission problem are modelled by different dielectricities ε_1 and ε_2.

L Shaped Domain

The Maxwell Eigenvalues $\lambda = \omega^2$ coincide with the non-zero Neumann Eigenvalues for the Laplace operator. Table 4.1 summarises the results of an overkill computation for the non-zero Neumann Eigenvalues [36]. Note that $\lambda_3 = \lambda_4 = \pi^2$.

λ_1	λ_2	λ_3	λ_4	λ_5
1.47562182408	3.53403136678	9.86960440109	9.86960440109	11.3894793979

Table 4.1: Non-zero Neumann Eigenvalues of the L shaped domain [36].

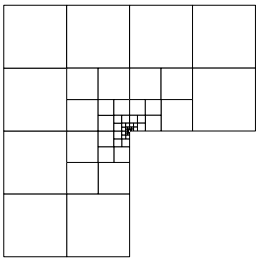


Figure 4.3: Geometric vertex mesh in the L shaped domain refined towards the reentrant corner.

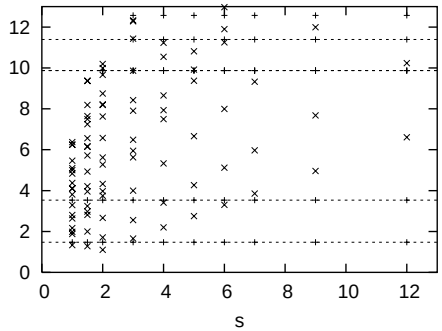


Figure 4.4: Eigenvalues in the L shaped domain (ordinate) plotted versus s (abscissa). For this plot, 15770 degrees of freedom are used. The horizontal dashed lines denote the exact Eigenvalues according to Table 4.1.

The first and fifth Eigenfunctions have a strong principal singularity which does not belong to $H^1(D)$ whereas the second belongs to $H^1(D)$ and the third and fourth Eigenfunction are even analytic in $\overline{D}$.

Figure 4.3 shows a geometric vertex mesh for the L shaped domain with refinement towards the reentrant corner $(0, 0)$. It is generated with the grading factor $\sigma = 1/2$ (*c. f.* Section 2.2).

Figure 4.4 shows a plot of the Eigenvalues on the ordinate versus different s values on the abscissa. $\times$ denotes spurious Eigenvalues, $+$ denotes physical Eigenvalues and $\circ$ denotes undecided (none in this case). The categorisation is done using the criterion (4.21). The Eigenvalues from Table 4.1 are indicated by horizontal dashed lines. It is clearly visible that the spurious Eigenvalues ($\times$) are lined up on straight lines through the origin of the plot whereas the physical Eigenvalues ($+$) are very close to the horizontal dashed lines.

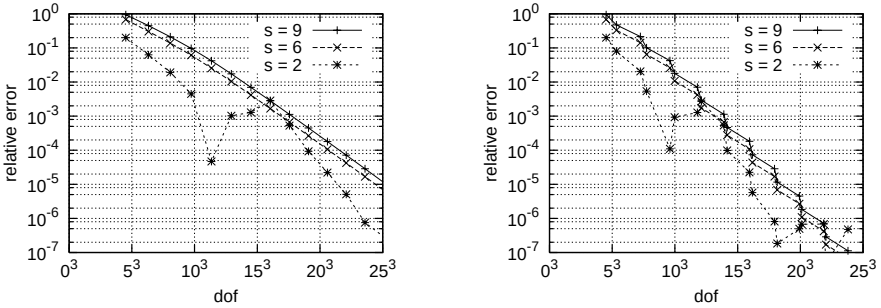


Figure 4.5: Convergence of the 1st Eigenvalue in the L shaped domain in log-$\sqrt[3]{\cdot}$ plot. The plots are done with $m = 1$ (left) and $m = 1/2$ (right).

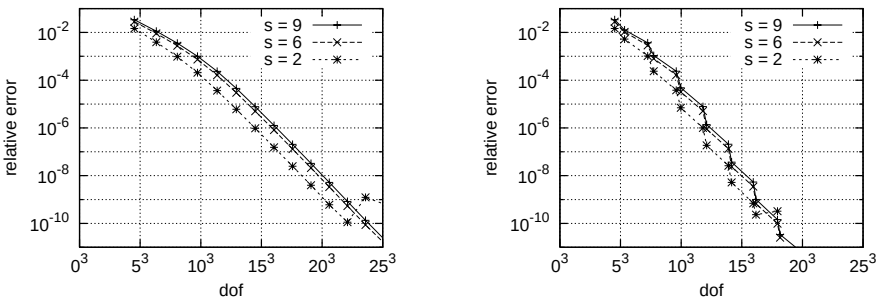


Figure 4.6: Convergence of the 2nd Eigenvalue in the L shaped domain in log-$\sqrt[3]{\cdot}$ plot. The plots are done with $m = 1$ (left) and $m = 1/2$ (right).

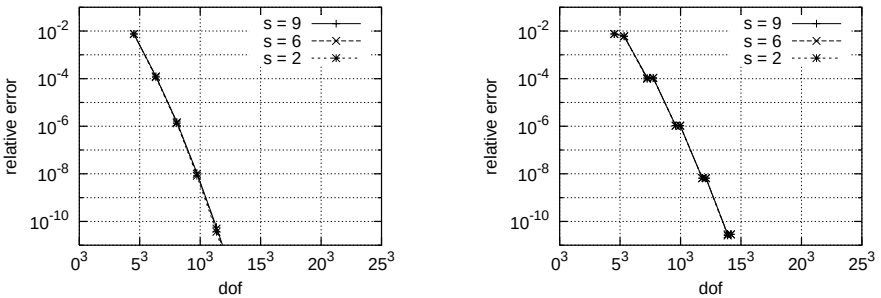


Figure 4.7: Convergence of the 3rd Eigenvalue in the L shaped domain in log-$\sqrt[3]{\cdot}$ plot. The plots are done with $m = 1$ (left) and $m = 1/2$ (right).

Figures 4.5–4.7 show the convergence history of the first three Eigenvalues for selected values of s. In each figure, there are two plots: the one on the left hand side with the classical refinement rule as introduced in Section 3.2 and the one on the right hand side with a modified refinement rule: only in every second step also the degrees were increased resulting in a polynomial degree distribution parameter $m = 1/2$. The first Eigenvalue (convergence plot in Figure 4.5) has a strong unbounded singularity. Nevertheless, the plot in log-$\sqrt[3]{\cdot}$ scale shows the anticipated exponential convergence. The second Eigenvalue (convergence plot in Figure 4.6) has its Eigenfunction in H^1. The convergence is also exponential but much faster. The third Eigenvalue (convergence plot in Figure 4.7, coinciding with the fourth Eigenvalue) has an analytic Eigenfunction. The convergence is still exponential in log-$\sqrt[3]{\cdot}$ scale but even faster.

It is observed that in Figures 4.5–4.6, the modified refinement rule wins over the classical refinement rule. However, in Figure 4.7 the classical refinement is seen to be more beneficial. It is well known that the optimal grading factor σ is close to 0.15 [63]. A geometrical hp-mesh with the modified refinement rule is closer to the optimal $\sigma = 0.15$ mesh than a mesh created with the classical refinement rule. The exception in the last case (Figure 4.7, convergence of the third Eigenvalue): the third and fourth Eigenfunction are known to be analytic. Analytic functions are best approximated with a p-extension. The classical rule is closer to the p-extension than the modified rule and therefore results in a faster convergence.

Transmission Problem

The transmission problem in the square $(-1, 1)^2$ (*c. f.* Figure 4.2) features a singularity in the centre of the domain and jumps of the normal components over the interfaces between the different materials M_1 and M_2. The materials M_1 and M_2 are modelled by different dielectricities $\varepsilon_1 \neq 1$ and $\varepsilon_2 = 1$. Below, we present the three cases $\varepsilon_1 = 1/2$, $\varepsilon_1 = 10^{-2}$ and $\varepsilon_1 = 10^{-8}$.

The same Eigenvalues as for the Maxwell Eigenproblem can be computed by the following Eigenproblem [36]:
Find frequencies ω and functions $0 \neq u \in H^1(D)$ such that

$$\int_D 1/\varepsilon \nabla u \cdot \nabla v \, d\boldsymbol{x} = \omega^2 \int_D uv \, d\boldsymbol{x} \quad \forall v \in H^1(D). \tag{4.22}$$

Using (4.22), the Eigenvalues $\lambda = \omega^2$ summarised in Table 4.2 can be computed.

	$\varepsilon_1 = 1/2$	$\varepsilon_1 = 10^{-2}$	$\varepsilon_1 = 10^{-8}$
λ_1	3.3175487634	4.89319332489	4.9348021587
λ_2	3.3663241572	7.20667542249	7.2252112326
λ_3	6.1863895624	15.53698165311	24.6740046478
λ_4	13.9263233310	24.46225024727	24.6740107936
λ_5	15.0829909612	24.48745601340	24.6740108178

Table 4.2: Non-zero Neumann Eigenvalues of the transmission problem with different ε_1 and $\varepsilon_2 = 1$, solved via (4.22) [36].

The use of the weighted regularisation in order to take the jumps into account cannot be validated, because the exponent γ of the weight w has to satisfy two incompatible conditions [37]:

- Compactness of $H_n \subset X_n[Y]$. In order that the number of spurious modes does not tend to infinity as the mesh is refined, $\gamma < 1$ is required.
- Density of $H_n \subset X_n[Y]$. For the transmission problem only, to ensure the approximation of the solutions, $\gamma > 1$ is required.

A possible solution is to use so-called *node doubling* at the interface and enforce the conditions between the different materials with additional constraints. These constraints are [70]:

- The tangential component of the electric field has to be continuous over the interface.
- The normal component of the electric field has to satisfy

$$(\boldsymbol{E}_l \cdot \boldsymbol{n}_l)\varepsilon_l + (\boldsymbol{E}_r \cdot \boldsymbol{n}_r)\varepsilon_r = 0, \tag{4.23}$$

  i. e. when $\varepsilon_l = \varepsilon_r$, the continuity of the normal component is enforced (ε_l and ε_r take the values of ε_1 and ε_2 depending on which interface we are looking at). Here, $\boldsymbol{n}_l$ and $\boldsymbol{n}_r$ are the outward unit normal vectors, and $\boldsymbol{E}_l$ and $\boldsymbol{E}_r$ are the electric field on the interface from the left and right respectively.

  At the origin of the domain, where the four subdomains meet, the conditions over the four interfaces for the normal and the tangential component should not be inforced individually[3] but as a sum [70].

[3] This would result in enforcing the Eigenfunction to be zero at the origin.

This Ansatz introduces a set of homogeneous, linear constraints to the Eigenvalue problem. A constrained matrix Eigenvalue problem can be solved using a QR factorisation of the matrix of constraints [62].

As before, plots of the Eigenvalues versus the scaling parameter s are shown. In these plots, $\times$ denotes spurious Eigenvalues, $+$ denotes physical Eigenvalues and $\circ$ denotes undecided. The categorisation is done using the criterion (4.21). Additionally, the Eigenvalues shown in Table 4.2 are indicated by horizontal dashed lines.

Basic Weighted Regularisation $\varepsilon_1 = 1/2$ is the easiest case with respect to the size of the jumps at the interfaces. We try to compute the first two Eigenvalues with basic weighted regularisation using the weight $w = \min(|x|, |y|)$ accomodating for the jumps at the interfaces and the (possible) singularities at the origin.

Figures 4.10–4.11 show the convergence history of the basic weighted regularisation method (without node doubling). Figure 4.8 shows the geometric edge mesh in the transmission domain $D = (-1, 1)^2$. The refinement towards the interfaces between the different materials should enable the method to resolve the jump in the normal component of the solution. The number of spurious Eigenvalues is—as predicted—unbounded for increasing number of mesh layers n (*c. f.* Figure 4.9). The convergence is not very good, however, the method seems to be able to find the correct Eigenvalues, at least in the pre-asymptotic range.

Weighted Regularisation with Node Doubling at the Interfaces We compute the first three Eigenvalues for the values of $\varepsilon_1 = 1/2, 10^{-2}$ and 10^{-8}. In addition to the results of the weighted regularisation, we present results of an edge element code[4] [3, 76] and the results of the equivalent problem (4.22) both on the same mesh[5] as the weighted regularisation computations (*c. f.* Figure 4.12).

Figure 4.13 shows the Eigenvalues computed with the weighted regularisation for the different values of ε_1. The number of spurious Eigenvalues is no longer unbounded and the spurious and physical Eigenvalues are separated more clearly. The plots in Figures 4.14–4.16 show the convergence histories for the first three Eigenvalues: The weighted regularisation with node doubling is able to resolve the Eigenvalues at an exponential rate of convergence—in most cases (see below). In

[4] The implementation of hp-edge elements in Concepts is due to Kersten Schmidt.

[5] This means we are using the same refinements and same polynomial degrees in all three cases. The spaces and the number of degrees of freedom N are different in all three cases.

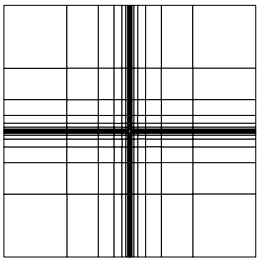


Figure 4.8: Geometric edge mesh towards the interfaces in the transmission domain $D = (-1, 1)^2$.

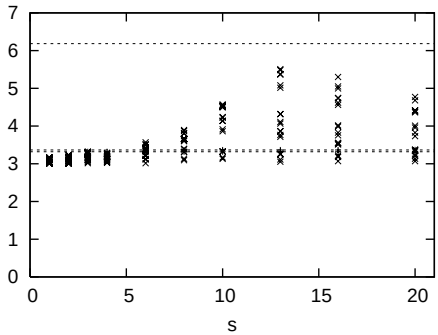


Figure 4.9: Weighted regularisation without node doubling in a geometric edge mesh (20766 degrees of freedom). Eigenvalues in the transmission domain $D = (-1, 1)^2$ (ordinate) plotted versus s (abscissa) for $\varepsilon_1 = 1/2$. The Eigensolver computes only Eigenvalues above 3 via the shift-invert method.

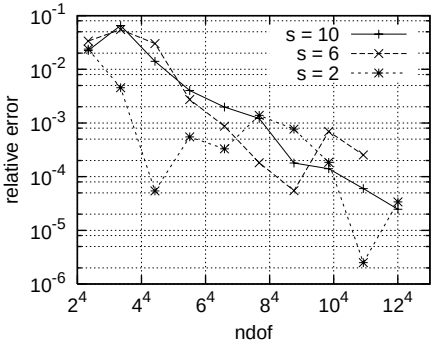


Figure 4.10: Convergence of the 1st Eigenvalue in the transmission domain with $\varepsilon_1 = 1/2$ in log-$\sqrt[4]{\cdot}$ scale computed with the weighted regularisation without node doubling at the interface.

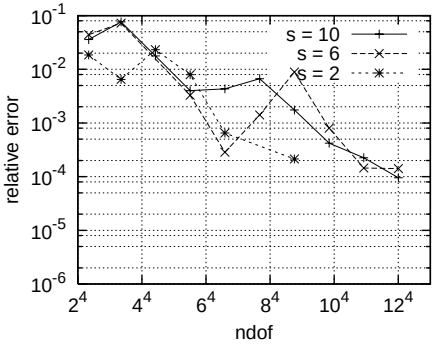


Figure 4.11: Convergence of the 2nd Eigenvalue in the transmission domain with $\varepsilon_1 = 1/2$ in log-$\sqrt[4]{\cdot}$ scale computed with the weighted regularisation without node doubling at the interface.

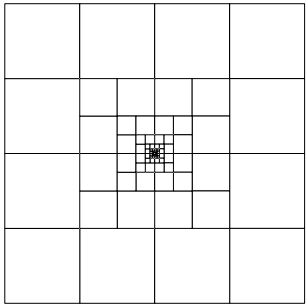


Figure 4.12: Geometric vertex mesh towards the origin in the transmission domain $D = (-1, 1)^2$.

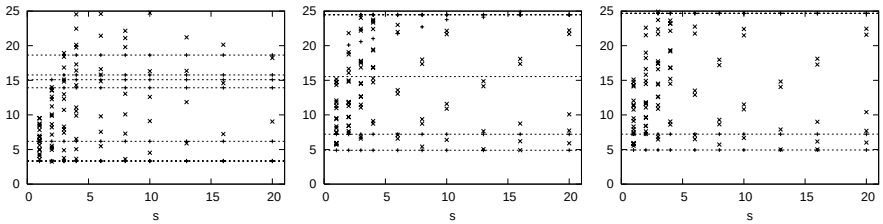


Figure 4.13: Eigenvalues in the transmission domain $D = (-1, 1)^2$ (ordinate) plotted versus s (abscissa) computed with node doubling at the interface for $\varepsilon_1 = 1/2$ (left), 10^{-2} (middle) and 10^{-8} (right). The horizontal dashed lines denote the exact Eigenvalues according to Table 4.2. 7680 degrees of freedom are used for these plots.

addition to the weighted regularisation, the plots also show the results of the edge element code and the equivalent problem (4.22).

Choice $\varepsilon_1 = 1/2$ All three methods show exponential convergence for the first three Eigenvalues in the left plots of Figures 4.14–4.16. The exponential rate of convergence of the weighted regularisation and the edge elements is the nearly the same for the first two Eigenvalues.

Choice $\varepsilon_1 = 10^{-2}$ The edge elements and the equivalent problem show exponential convergence for the first three Eigenvalues in the middle plots of Fig-

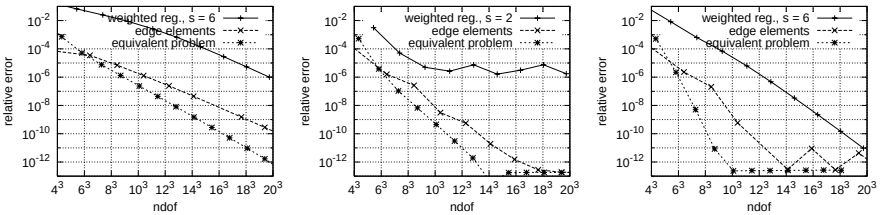


Figure 4.14: Convergence history of the 1st Eigenvalue for $\varepsilon_1 = 1/2$ (left), 10^{-2} (middle) and 10^{-8} (right). The results of the weighted regularisation with node doubling, the edge elements and the equivalent problem all show exponential convergence in the $\log\sqrt[3]{\cdot}$ scale plots. The problems for $\varepsilon_1 = 10^{-2}$ are described in the text.

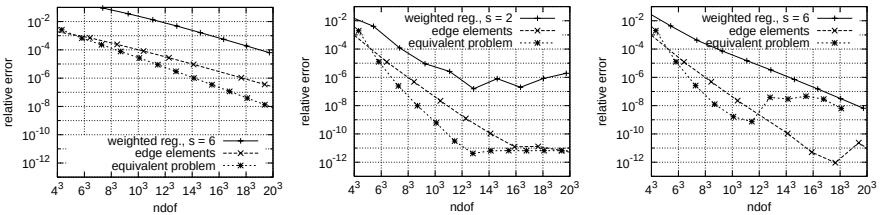


Figure 4.15: Convergence history of the 2nd Eigenvalue for $\varepsilon_1 = 1/2$ (left), 10^{-2} (middle) and 10^{-8} (right). The results of the weighted regularisation with node doubling, the edge elements and the equivalent problem all show exponential convergence in the $\log\sqrt[3]{\cdot}$ scale plots. The problems for $\varepsilon_1 = 10^{-2}$ and 10^{-8} are described in the text.

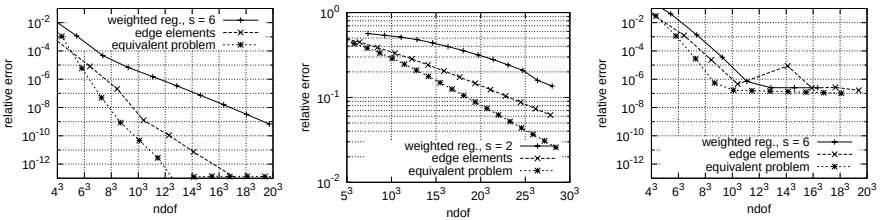


Figure 4.16: Convergence history of the 3rd Eigenvalue for $\varepsilon_1 = 1/2$ (left), 10^{-2} (middle) and 10^{-8} (right). The results of the weighted regularisation with node doubling, the edge elements and the equivalent problem all show exponential convergence in the $\log\sqrt[3]{\cdot}$ scale plots. The problems for $\varepsilon_1 = 10^{-8}$ are described in the text. Note the different scales for in the plot for $\varepsilon_1 = 10^{-2}$ (middle).

ures 4.14–4.16. The weighted regularisation only does so for the third Eigenvalue. The convergence histories for the first and second Eigenvalue deteriorate at a level of the relative error of 10^{-6}. This does not depend on s, the chosen Eigenvalue solver (or its parameters)[6] or the formulation of the constraints (4.23) at the origin[7].

The middle plot in Figure 4.16 shows the convergence history for the third Eigenvalue: It is approximated poorly. The reason is that the third Eigenfunction has a strong unbounded singularity at the origin.[8] This can also be seen in Figure 4.13(b): The Eigenvalues computed with the weighted regularisation are above 20 for $s \geq 2$ and even above 25 for $s \geq 16$—the true value is expected to be close to 15.

Choice $\varepsilon_1 = 10^{-8}$ All three methods show exponential convergence for the first three Eigenvalues in the right plots of Figures 4.14–4.16 with the following two exceptions:

- The equivalent problem (4.23) is only able to resolve the second Eigenvalue up to a relative precision of 10^{-7}. This has been confirmed for different meshes, refinements and parameters for the Eigensolver.

- The third Eigenvalue can only be resolved up to a relative precision of 10^{-7}. The reason is that the third and fourth Eigenvalue[9] are very close:

$$\frac{\lambda_4 - \lambda_3}{\lambda_4} \approx 10^{-7},$$

  this results in an ill-conditioned matrix Eigenproblem [22]. The Eigensolver ARPACK [78, 77] has problems yielding high accuracy under these circumstances.

[6] We normally use ARPACK [77, 78] with Umfpack as linear solver [38, 39, 40] but also tried the Eigensolver JDBSYM [59] and the linear solvers Pardiso[91] and SuperLU [45]. The shift parameter for the shift-invert method and the number of Eigenvalues to be computed were varied.

[7] Tested variants are: individually over each interface, as a sum or no constraints at the origin at all.

[8] In the case of the equivalent problem (4.22), the third Eigenfunction has a jump between the two subdomains with ε_1.

[9] In fact, the fourth Eigenvalue is double: $\lambda_4 = \lambda_5$.

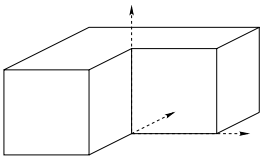


Figure 4.17: The thick L shaped domain is the Cartesian product of the two dimensional L shaped domain $(-1,1)^2 \setminus (0,1)\times(-1,0)$ with $(0,1)$ in the third direction.

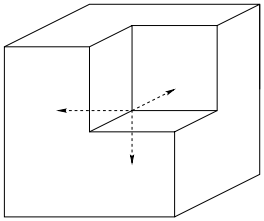


Figure 4.18: Fichera corner $(-1,1)^3 \setminus (-1,0)^3$.

4.3.3 Three Dimensions

The thick L shaped domain (Figure 4.17), the Fichera corner (Figure 4.18) and the double Fichera corner are treated. The thick L shaped domain features a singularity at the reentrant edge whereas the Fichera corner produces singular solutions at all three reentrant edges and the reentrant corner. The double Fichera corner has two cubes cut out opposite each other where the Fichera corner has only one. The open domain D of the double Fichera corner is not simply connected and it features a so-called *topological singularity* which behaves like $1/r$ at the origin.

The Thick L Shaped Domain

The reference solutions for the first few Eigenvalues $\lambda = \omega^2$ of (4.8) are

1. either the sum of a non-zero Neumann Eigenvalue (NA1, NA2, ...) in the two dimensional L shaped domain and a Dirichlet Eigenvalue (DJ1, DJ2, ...) in $(0,1)$ (*i. e.* $\pi^2, 4\pi^2, \ldots$),

2. or the sum of a Dirichlet Eigenvalue (DA1, DA2, ...) in the two dimensional L shaped domain and a Neumann Eigenvalue (NJ0, NJ1, ...) in $(0,1)$ (*i. e.* $0, \pi^2, \ldots$).

The numerical values of the first nine Eigenvalues are shown in Table 4.3.

Figure 4.19 shows geometric edge meshes for the thick L shaped domain. The left hand mesh is generated with the grading factor $\sigma = 1/2$. It is essentially a

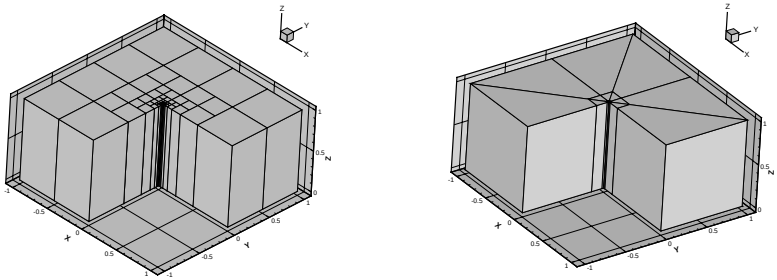


Figure 4.19: Geometric edge meshes in the thick L shaped domain refined towards the reentrant edge. The meshes on the left and right have grading factors $\sigma = 1/2$ and $\sigma = 0.15$ respectively.

tensor product of a two dimensional mesh for an L shaped domain with geometric refinement towards the reentrant corner (*c. f.* Figure 4.3) and a simple one-element mesh in the third direction. It is treated and built as a three dimensional mesh in Concepts, though using the algorithm introduced in Section 3.2. The right hand mesh in Figure 4.19 shows a geometric mesh with grading factor $\sigma = 0.15$. For each number of layers, it is created using a hand written mesh generator.

Figure 4.20 shows a plot of the Eigenvalues on the ordinate versus different s values on the abscissa. $\times$ denotes spurious Eigenvalues, $+$ denotes physical Eigenvalues and $\circ$ denotes undecided. The categorisation is done using the criterion (4.21). The Eigenvalues from Table 4.3 are indicated by horizontal dashed lines. It is clearly visible that the spurious Eigenvalues ($\times$) are lined up on straight lines through the origin of the plot whereas the physical Eigenvalues ($+$) are very close to the horizontal dashed lines.

Figures 4.21–4.23 show the convergence history of the first three Eigenvalues for selected values of s. In each figure, there are two plots: the one on the left hand side with $\sigma = 1/2$ and the one on the right hand side with $\sigma = 0.15$. Figures 4.24–4.26 show the convergence history of the first three Eigenvalues for selected values of s. In each figure, there are two plots: the one on the left hand side with $\sigma = 1/8$ and the one on the right hand side with $\sigma = 1/10$. The second Eigenvalue shown in Figure 4.22 has a strong singularity at the reentrant edge, the first and third are in H^1 and could therefore also be captured by a method with a constant weight

$\lambda_1 =$ DA1 + NJ0	$\lambda_2 =$ NA1 + DJ1
9.6397238447	11.3452262252
$\lambda_3 =$ NA2 + DJ1	$\lambda_4 =$ DA2 + NJ0
13.4036357679	15.1972519265
$\lambda_5 =$ DA1 + NJ1	$\lambda_6 =$ DA3 + NJ0
19.5093282458	19.7392088022
$\lambda_7 =$ NA3 + DJ1	$\lambda_8 =$ NA4 + DJ1
19.7392088022	19.7392088022
$\lambda_9 =$ NA5 + DJ1	
21.2590837990	

Table 4.3: Numeric values of the first nine Eigenvalues in the thick L shaped domain.

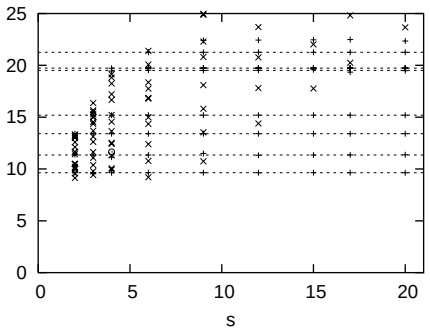


Figure 4.20: Eigenvalues in the thick L shaped domain (ordinate) plotted versus s (abscissa). For this plot, 18276 degrees of freedom are used. The horizontal dashed lines denote the exact Eigenvalues.

(*i. e.* $\gamma = 0$). None of these Eigenvalues are multiple. All plots are in log-$\sqrt[4]{\cdot}$ scale and show the anticipated straight lines. This gives experimental evidence for the conjectured exponential convergence of Maxwell Eigenvalues in polyhedra. Similarly to the two dimensional L shaped domain, the meshes with $\sigma = 0.15$ give test best convergence results out of the four geometric grading parameters.

Fichera Corner

Figure 4.27 shows a geometric edge mesh for the Fichera corner. Note, that this is not a tensor product type mesh. These refinements (with the many small elements) do not reach far into the domain and are not visible 'from the other side' of the domain.

As seen in the case of the thick L shaped domain, Figure 4.28 shows a plot of the Eigenvalues on the ordinate versus the s values on the abscissa. Again, one can see that the spurious Eigenvalues are lined up on straight lines through the origin in the s vs. Eigenvalue plot.

Figures 4.29–4.31 show the convergence history of the first three Eigenvalues for selected values of s. In each figure, there are two plots: the one on the left hand

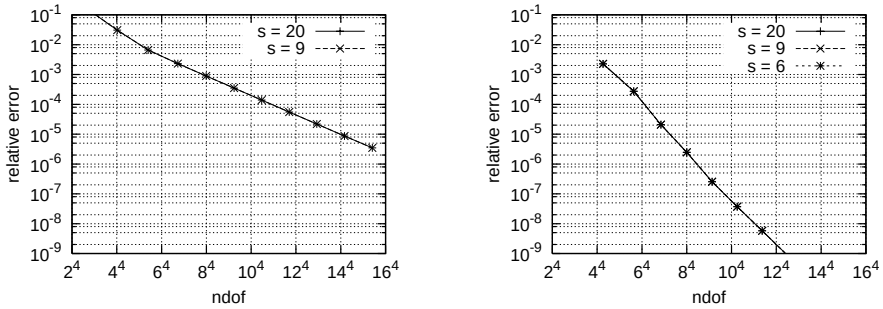


Figure 4.21: Convergence of the 1st Eigenvalue in the thick L shaped domain in log-$\sqrt[4]{\cdot}$ plot. The left and right plots were done with $\sigma = 1/2$ and $\sigma = 0.15$ respectively. Note that the convergence histories are identical for all three values of s.

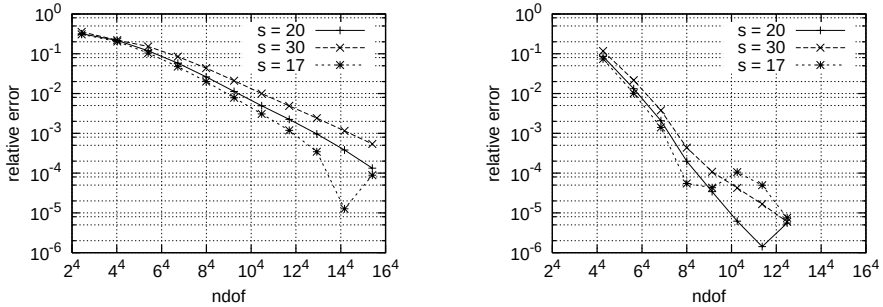


Figure 4.22: Convergence of the 2nd Eigenvalue in the thick L shaped domain in log-$\sqrt[4]{\cdot}$ plot. The left and right plots were done with $\sigma = 1/2$ and $\sigma = 0.15$ respectively.

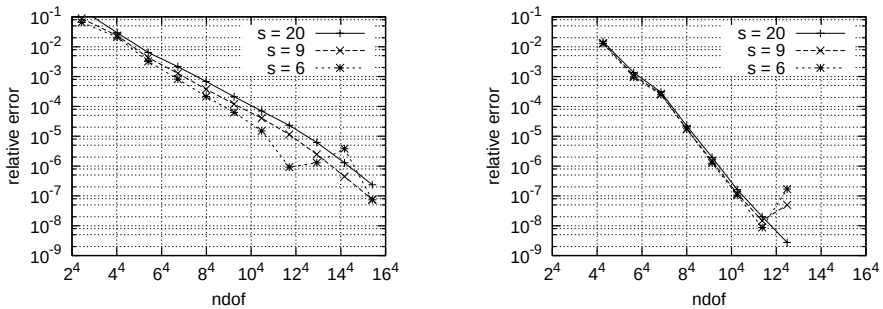


Figure 4.23: Convergence of the 3rd Eigenvalue in the thick L shaped domain in log-$\sqrt[4]{\cdot}$ plot. The left and right plots were done with $\sigma = 1/2$ and $\sigma = 0.15$ respectively.

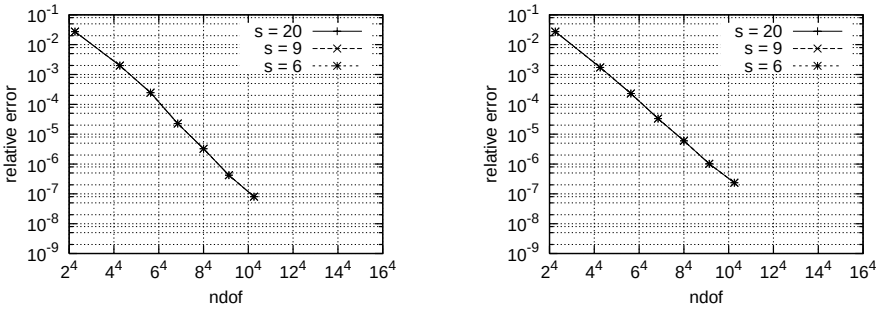


Figure 4.24: Convergence of the 1st Eigenvalue in the thick L shaped domain in log-$\sqrt[4]{\cdot}$ plot. The left and right plots were done with $\sigma = 1/8$ and $\sigma = 1/10$ respectively. Note that the convergence histories are identical for all three values of s.

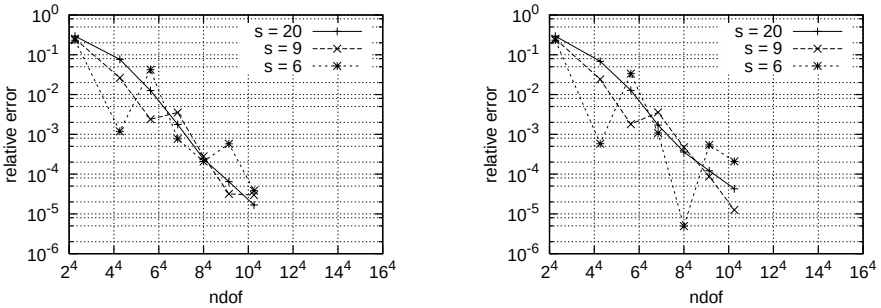


Figure 4.25: Convergence of the 2nd Eigenvalue in the thick L shaped domain in log-$\sqrt[4]{\cdot}$ plot. The left and right plots were done with $\sigma = 1/8$ and $\sigma = 1/10$ respectively.

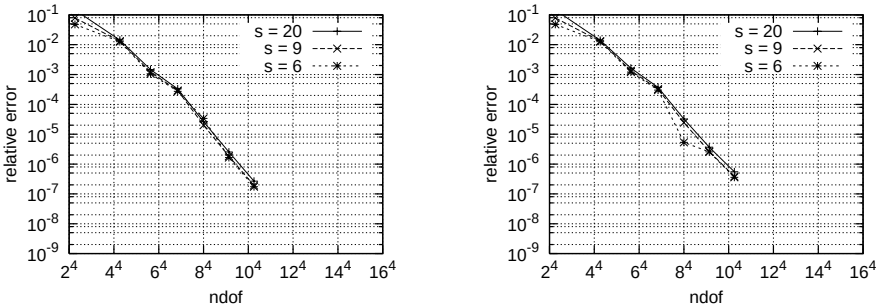


Figure 4.26: Convergence of the 3rd Eigenvalue in the thick L shaped domain in log-$\sqrt[4]{\cdot}$ plot. The left and right plots were done with $\sigma = 1/8$ and $\sigma = 1/10$ respectively.

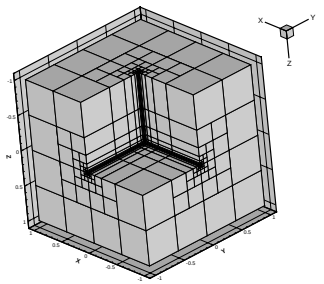


Figure 4.27: Geometric edge mesh in the Fichera corner refined towards reentrant edges and corner.

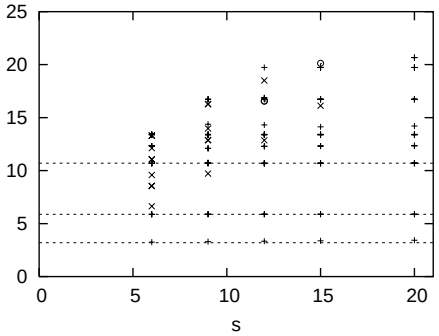


Figure 4.28: Eigenvalues in the Fichera corner (ordinate) plotted versus s (abscissa). $\times$ denotes spurious Eigenvalues, $+$ denotes physical Eigenvalues and $\circ$ denotes undecided. For this plot, 45102 degrees of freedom are used.

side with the simplified weight (4.7)

$$w = \operatorname{dist}\Big(\boldsymbol{x}, \bigcup_{c\in\mathcal{C}} c \cup \bigcup_{e\in\mathcal{E}} \overline{e}\Big)$$

and the one on the right with the weight (4.18)

$$w = \prod_{\boldsymbol{c}\in\mathcal{C}} r_{\boldsymbol{c}} \cdot \prod_{e\in\mathcal{E}} \rho_e.$$

The second Eigenvalue shown in Figure 4.30 is a double Eigenvalue and the next shown in Figure 4.31 is suspected to be either a double or even a triple Eigenvalue. All these plots are in log-$\sqrt[5]{\cdot}$ scale and show the anticipated straight lines. This gives experimental evidence for the conjectured exponential convergence of Maxwell Eigenvalues in polyhedra. As no exact values for the Eigenvalues in the Fichera corner are known, extrapolated values were used to generate the convergence graphs. These values are shown in Table 4.4. Comparing the left and right plots in Figures 4.29–4.31 reveals only slight differences in the convergence rates for the weights (4.7) and (4.18).

Figures 4.32–4.34 show the convergence history of the first three Eigenvalues for selected values of s for h- and p-extensions as opposed to hp-extensions which

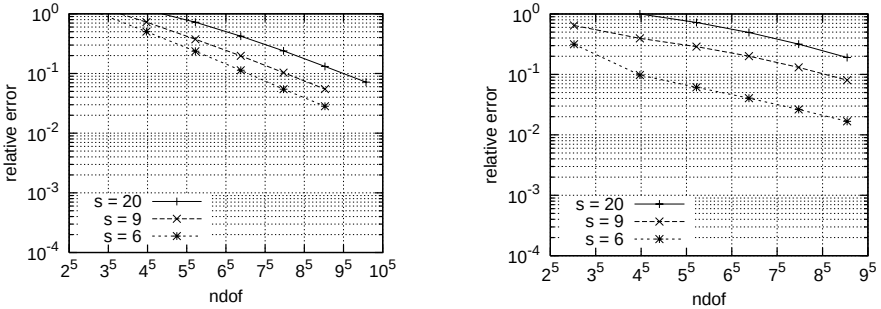


Figure 4.29: Convergence of the 1st Eigenvalue in the Fichera corner in log-$\sqrt[5]{\cdot}$ plot. The left and right plots were done using the weights w (4.7) and (4.18) ($\gamma_{\max} = 1$) respectively.

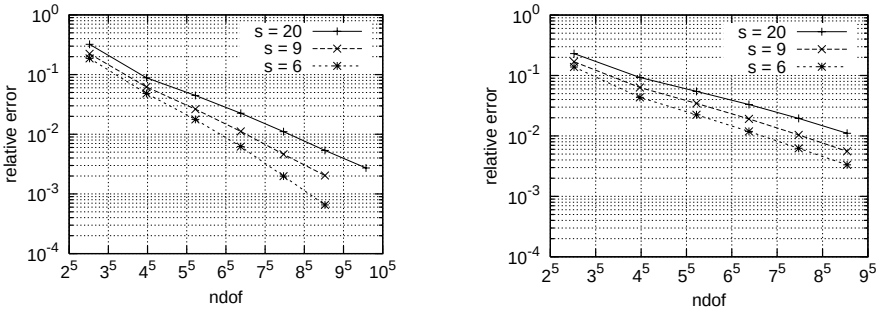


Figure 4.30: Convergence of the 2nd Eigenvalue in the Fichera corner in log-$\sqrt[5]{\cdot}$ plot. The left and right plots were done using the weights w (4.7) and (4.18) ($\gamma_{\max} = 1$) respectively.

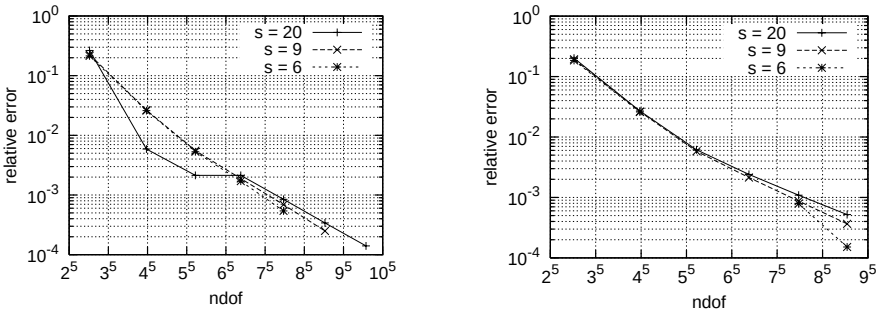


Figure 4.31: Convergence of the 3rd Eigenvalue in the Fichera corner in log-$\sqrt[5]{\cdot}$ plot. The left and right plots were done using the weights w (4.7) and (4.18) ($\gamma_{\max} = 1$) respectively.

extrapolated Eigenvalue	3.197	5.878	10.6955
reliable digits	2	4	4

Table 4.4: Extrapolated Eigenvalues for the Fichera corner

extrapolated Eigenvalue	6.82	8.806	9.72
reliable digits	2	3	2

Table 4.5: Extrapolated Eigenvalues for the double Fichera corner

were shown so far. In each figure, there are two plots: the one on the left hand side with h-extensions and the one on the right hand side with p-extensions. The chosen weight was the simplified weight (4.7). In the h-extension case, the degree of all elements was $p = 2$ and all refinements were uniform. In the p-extension case, $7 \cdot 8^2 = 448$ elements were used and the degree was uniform and isotropic in the whole domain D. Two (expected) conclusion can be drawn from these results:

- The hp-extensions (Figures 4.29–4.31) give a higher precision than h- or p-refinement at the same number of degrees of freedom.
- Neither the h- nor the p-extensions can deliver exponential convergence like the hp-extensions.

4.3.4 Double Fichera Corner

The computations in the double Fichera corner are done with the simplified weight (4.7), $\gamma_{\max} = 1$. The mesh is created by marking the origin and the reentrant edges using Algorithm 3.3.

Figure 4.35 shows a plot of the Eigenvalues on the ordinate versus the s values on the abscissa. As before, the spurious Eigenvalues are lined up on straight lines through the origin in the s vs. Eigenvalue plot and the physical Eigenvalues are on horizontal lines. The horizontal, dashed lines have the values given in Table 4.5. The third and fourth Eigenvalues are double ($\lambda_3 = \lambda_4$ and $\lambda_5 = \lambda_6$).

Figures 4.36–4.38 show the convergence histories of the first three Eigenvalues for selected values of s. As before, we can observe a non-monotone convergence behaviour for the smallest value of s shown here ($s = 6$), *c. f.* Section 4.3.1.

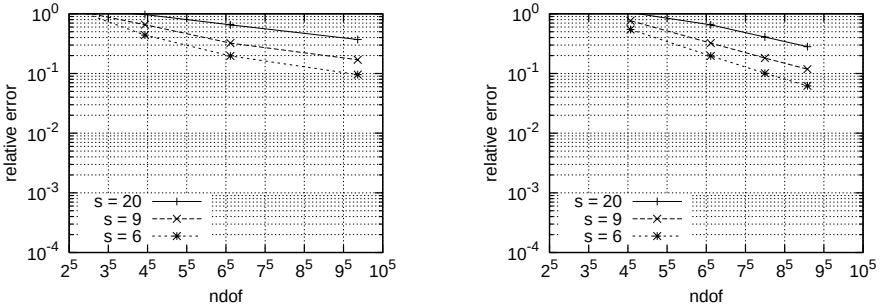


Figure 4.32: Convergence of the 1st Eigenvalue in the Fichera corner in log-$\sqrt[5]{\cdot}$ plot. The left and right plots were done using uniform h- and p-extensions respectively.

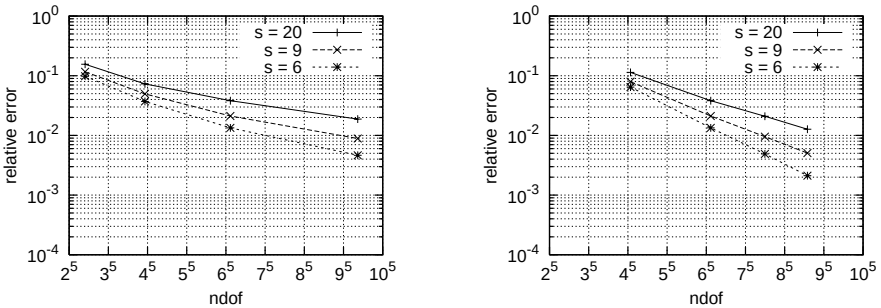


Figure 4.33: Convergence of the 2nd Eigenvalue in the Fichera corner in log-$\sqrt[5]{\cdot}$ plot. The left and right plots were done using uniform h- and p-extensions respectively.

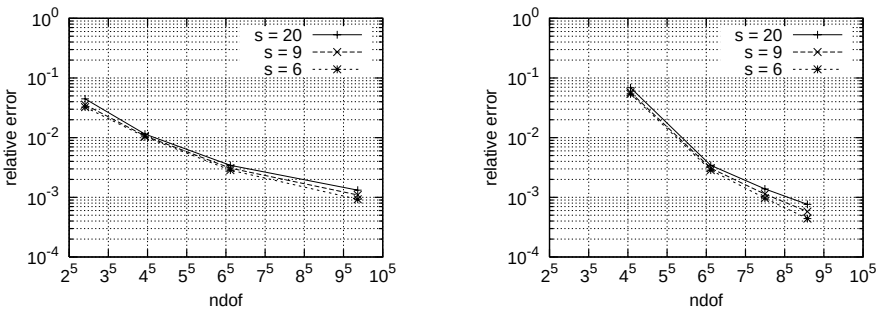


Figure 4.34: Convergence of the 3rd Eigenvalue in the Fichera corner in log-$\sqrt[5]{\cdot}$ plot. The left and right plots were done using uniform h- and p-extensions respectively.

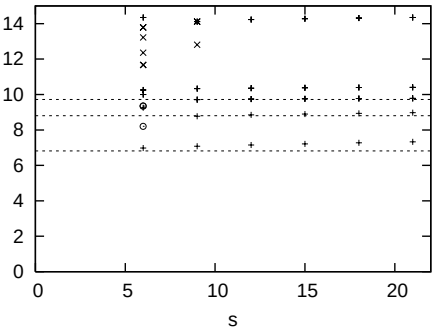


Figure 4.35: Eigenvalues in the double Fichera corner (ordinate) plotted versus s (abscissa). $\times$ denotes spurious Eigenvalues, $+$ denotes physical Eigenvalues and $\circ$ denotes undecided. For this plot, 61920 degrees of freedom are used.

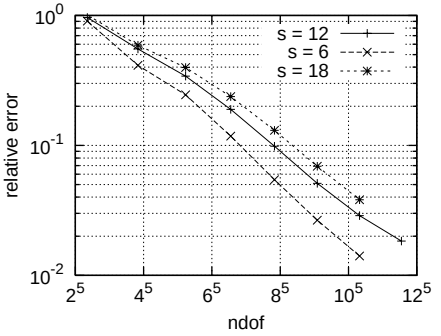


Figure 4.36: Convergence of the 1st Eigenvalue in the double Fichera corner in log-$\sqrt[5]{\cdot}$ plot.

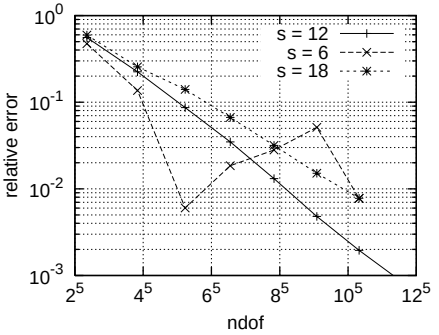


Figure 4.37: Convergence of the 3rd Eigenvalue in the double Fichera corner in log-$\sqrt[5]{\cdot}$ plot.

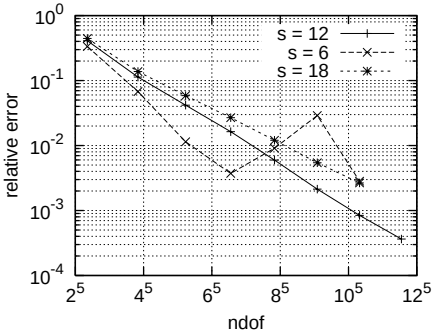


Figure 4.38: Convergence of the 3rd Eigenvalue in the double Fichera corner in log-$\sqrt[5]{\cdot}$ plot.

4.3.5 Conclusion

The weighted regularisation is able to deliver the conjectured exponential convergence in the two and three dimensional examples with constant coefficients considered. The singularities introduced by the reentrant corners and edges in the given domains are resolved as predicted.

However, the weighted regularisation fails to deliver exponential convergence in some cases of the two dimensional example with jumping coefficients. The instability of the weighted regularisation is not systematic. In the example with the jumping coefficients, the weighted regularisation is compared to the 'classical' discretisation method for Maxwell's equations (Nédélec's edge elements): the edge elements yield exponential convergence on the same mesh-degree combination as the weighted regularisation.

Experience with Scaling Parameter s

The equation (4.20) solved in the numerical examples contains a scaling parameter s not present in the formulation (4.8). We discuss our numerical experience in the *selection of the parameter s* and its impact in the presence of spurious Eigenvalues.

Larger values of s move the spurious Eigenvalues further up while they increase the gap to the exact value.

Consider the Eigenvalues versus s plots. The lining-up on straight lines through the origin of the *spurious Eigenvalues* is good—already for moderate numbers of degrees of freedom. However, only in the asymptotic limit, the *physical Eigenvalues* are lined up on horizontal lines. The plot in the middle of Figure 4.16 shows this most clearly (and can also be observed in all other examples). The numerical values for the third Eigenvalue (15.5...) are found on a slightly curved line between ($s = 2, \lambda = 20$) and ($s = 13, \lambda = 25$). Increasing the number of degrees of freedom brings this curve more and more down to the exact value.

5 Elliptic Partial Differential Equations with Stochastic Coefficients

If one is able to control the *discretisation error*, and assuming that the *modelling error* inherent in the selected partial differential equations is negligible (*i. e.* that the adopted PDEs precisely describe the physics of the system under consideration), the gap that remains between simulation and observation must be due to uncertainty in the input data.

This requires new developments in several areas of applied mathematics and engineering: input parameters are to be replaced by random variables, whose *statistics* must be estimated, and the governing PDEs must be reformulated as stochastic PDEs. Traditional deterministic Finite Element solutions must be reformulated to allow for randomness in input data and solution.

This chapter focuses on this latter aspect, *i. e.* the formulation, design and implementation of *deterministic* Finite Element-based solution methods to stochastic elliptic PDEs. More theory and the related proofs can be found in [99, 107].

After a short introduction in the first section, the Karhunen-Loève expansion of the stochastic diffusion coefficients to separate stochastic and physical variables is introduced in the second section. The third section gives a detailed description of the stochastic Galerkin method. The fourth and fifth sections carefully explain the two key computational tasks: the fast computation of the Karhunen-Loève expansion and the parallel solution of the deterministic problems, respectively. Finally, the last section shows some numerical examples.

5.1 Introduction and Problem Formulation

The model problem is a stationary elliptic diffusion problem in a domain D with stochastic diffusion coefficient $a(\boldsymbol{x}, \omega)$ assumed for simplicity to be isotropic. To specify assumptions on the coefficients $a(\boldsymbol{x}, \omega)$, let (Ω, Σ, P) be a σ-finite probability space and $D \subset \mathbb{R}^d$ a bounded open set with Lipschitz boundary $\Gamma = \partial D$. Assume that $a \in L^\infty(D \times \Omega)$ is strictly positive, with lower and upper bound α and β respectively,

$$0 < \alpha \leq a(\boldsymbol{x}, \omega) \leq \beta < \infty, \quad \lambda \times P\text{-a. e. } (\boldsymbol{x}, \omega) \in D \times \Omega, \tag{5.1}$$

where λ is the Lebesgue measure in $\mathbb{R}^d$. (5.1) implies $a \in L^2(\Omega, dP; L^\infty(D))$.

Consider the following model problem, with stochastic left hand side,

$$\begin{aligned} -\operatorname{div}(a(\boldsymbol{x}, \omega)\nabla_x u(\boldsymbol{x}, \omega)) &= f(\boldsymbol{x}) \quad \text{in } D \\ u(\boldsymbol{x}, \omega) &= 0 \quad \text{on } \partial D \end{aligned} \qquad P\text{-a. e. } \omega \in \Omega. \tag{5.2}$$

The coefficient $a(\boldsymbol{x}, \omega)$ as well as the solution $u(\boldsymbol{x}, \omega)$ are random fields in $D \subset \mathbb{R}^d$, *i. e.* they are random variables $X \in (\Omega, \Sigma, P)$ in every physical location $\boldsymbol{x} \in D$. a and u are both jointly measurable functions from $D \times \Omega$ to $\mathbb{R}$.

Assume that the known information about the diffusion coefficient a includes its mean field and its two-point correlation, given by

$$E_a(\boldsymbol{x}) := \int_\Omega a(\boldsymbol{x}, \omega)\, dP(\omega) \quad \text{and} \quad C_a(\boldsymbol{x}, \boldsymbol{x}') := \int_\Omega a(\boldsymbol{x}, \omega) \cdot a(\boldsymbol{x}', \omega)\, dP(\omega),$$

i. e. that $E_a(\boldsymbol{x})$ and $C_a(\boldsymbol{x}, \boldsymbol{x}')$ are explicitly and exactly known.[1] Equivalently, the covariance V_a could be given, since

$$V_a(\boldsymbol{x}, \boldsymbol{x}') = C_a(\boldsymbol{x}, \boldsymbol{x}') - E_a(\boldsymbol{x})E_a(\boldsymbol{x}').$$

For C_a, V_a to exist, $a(\boldsymbol{x}, \omega)$ must have finite second moments (which is assured by (5.1)).

Given this information on $a(\boldsymbol{x}, \omega)$ and a known deterministic source term $f(\boldsymbol{x})$ (this could be relaxed as well, see *e. g.* [99, 100]; but here, our aim is to solve

[1] This is a rather optimistic assumption since often a functional form of $C_a(\boldsymbol{x}, \boldsymbol{x}')$ is postulated with a finite number of free parameters which are statistically estimated from the available data. However, this problem has to be solved on the modelling side.

(5.2)), $u(\boldsymbol{x}, \omega)$ is a mathematically well-defined object. However, the task 'compute $u(\boldsymbol{x}, \omega)$' is less obvious to realize numerically and of limited interest in practice. In applications, only certain statistics and moments of $u(\boldsymbol{x}, \omega)$ are of interest, and this is also our goal of computation: *given statistics E_a and C_a of the data, compute statistics of the random solution u*, like E_u, C_u or probabilistic level sets,

$$D_\varepsilon^\delta := \{\boldsymbol{x} \in D : P(|u(\boldsymbol{x}, \cdot)| > \delta) < \varepsilon\}\,.$$

Overview of Numerical Methods

Monte Carlo Method

The simplest approach to a numerical solution of (5.2) is Monte Carlo simulation. There is a vast choice of literature on this subject of which only [66, 71] are mentioned. This means to generate numerous samples of $a(\boldsymbol{x}, \omega)$ with prescribed statistics, solve (5.2) for each sample, and to determine the statistics of $u(\boldsymbol{x}, \omega)$ from the set of solutions. Due to the generally slow convergence of Monte Carlo methods, this approach requires a rather large number of 'samples', *i. e.* a large number of solutions of deterministic boundary value problems. Conceptually, Monte Carlo simulation corresponds to a 'collocation in ω'.

Perturbation Methods

Perturbation methods (see *e. g.* [17]) to solve (5.2) represent the stochastic solution as an exponentially convergent infinite series, in which each term solves a problem with the same deterministic coefficient (that is, independent of ω) but different stochastic loadings. It turns out that in order to compute exactly even the simplest statistic of u, namely E_u, one has to know the distribution function of $a(\boldsymbol{x}, \cdot)$ at any $\boldsymbol{x} \in D$—a very strong requirement.

Stochastic Galerkin Methods

In this chapter, we outline a stochastic Galerkin method [55] for the numerical solution of (5.2) which can be understood as Galerkin discretisation in probability space. The idea of reducing a stochastic equation to a large system of deterministic ones is not new—stochastic Galerkin methods have attracted considerable attention in recent years, we mention here only [60] and the references therein.

Unlike the 'collocation' type Monte Carlo approaches, in stochastic Galerkin Finite Element Methods the stochastic 'variable' ω is discretised by an orthogonal projection with respect to the probability measure P onto a finite dimensional subspace of (Ω, Σ, P). The probability space being infinite dimensional, the feasibility of a stochastic Galerkin discretisation of (5.2) strongly depends on the availability of a good basis of $L^2(\Omega, dP)$. In numerous works [60, 113], the use of a so-called *Wiener Chaos expansion* [110, 111] has been advocated. Here, a *Karhunen-Loève* [80, 81] expansion of the random field $a(\boldsymbol{x}, \omega)$ is used to generate coordinates with certain optimality conditions for the deterministic approximation of the random solution. This is made possible by a kernel independent Fast Multipole Method [93, 94] to compute the Eigenpairs of the covariance operator for $a(\boldsymbol{x}, \omega)$ in log-linear complexity per Eigenpair.

The stochastic Galerkin FEM is, like the deterministic FEM, based on a variational formulation of (5.2). To define it, introduce the Hilbert space $\mathcal{H}_0^1(D)$ of $H_0^1(D)$-valued random fields with finite second moments

$$\mathcal{H}_0^1(D) := L^2(\Omega, dP; H_0^1(D)).$$

Then, the variational form of (5.2) reads
Find $u \in \mathcal{H}_0^1(D)$ such that for every $v \in \mathcal{H}_0^1(D)$

$$\int_\Omega \left(\int_D a(\boldsymbol{x}, \omega) \nabla_x u \cdot \nabla_x v \, d\boldsymbol{x} \right) dP(\omega) = \int_\Omega \left(\int_D f(\boldsymbol{x}, \omega) v(\boldsymbol{x}, \omega) \, d\boldsymbol{x} \right) dP(\omega). \tag{5.3}$$

We conclude this section by mentioning that under assumption (5.1), the existence and uniqueness of a solution u to (5.3) follows from the Lax-Milgram Lemma.

5.2 Karhunen-Loève Expansion

To reduce (5.2) to a deterministic (albeit infinite dimensional) problem, deterministic and stochastic variables in the coefficient $a(\boldsymbol{x}, \omega)$ are separated. The theoretical tool required to achieve this is the so-called Karhunen-Loève expansion.

Let the random diffusion coefficient $a \in L^2(D \times \Omega)$, then $V_a \in L^2(D \times D)$ and its *covariance operator*

$$\mathcal{V}_a : L^2(D) \to L^2(D), \quad (\mathcal{V}_a u)(\boldsymbol{x}) := \int_D V_a(\boldsymbol{x}, \boldsymbol{x}') u(\boldsymbol{x}') \, d\boldsymbol{x}' \quad \forall u \in L^2(D)$$

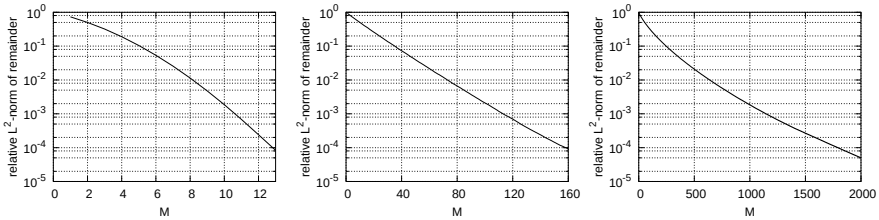


Figure 5.1: Convergence rates of the Karhunen-Loève series in one (left plot), two (middle) and three dimensions (right). All plots show the relative decay of the Karhunen-Loève series remainder after truncation at level M in the L^2-norm plotted against the truncation parameter M. The kernel is $\exp(-10|\boldsymbol{x} - \boldsymbol{x}'|^2)$ and the domain D is the unit box $(-1, 1)^d$.

is a symmetric, positive semi-definite and compact integral operator. Therefore, it has a countable sequence $\{\lambda_m, \varphi_m\}_{m\geq 1}$ of Eigenpairs with real, bounded Eigenvalues $\lambda_1 \geq \lambda_2 \geq \ldots$ with $\lambda_m \to 0$ as $m \to \infty$.

Moreover, there exists a sequence of random variables $\{X_m\}_{m\geq 1}$ such that

$$\int_\Omega X_m(\omega)\, dP(\omega) = 0 \text{ and } \int_\Omega X_n(\omega) X_m(\omega)\, dP(\omega) = \delta_{nm} \quad \forall n, m \geq 1,$$

and $a(\boldsymbol{x}, \omega)$ can be expanded in a *Karhunen-Loève expansion:*

$$a(\boldsymbol{x}, \omega) = E_a(\boldsymbol{x}) + \sum_{m\geq 1} \sqrt{\lambda_m}\varphi_m(\boldsymbol{x}) X_m(\omega). \tag{5.4}$$

Remark 5.5 (Convergence of the Karhunen-Loève expansion)
In (5.4)*, the Karhunen-Loève series converges in $L^2(D \times \Omega)$ at the same rate as the sum of Eigenvalues, c. f. Figure 5.1.*

Additionally, if the sequences $\{\varphi_m\}_{m\geq 1}$ and $\{X_m\}_{m\geq 1}$ are uniformly bounded in $L^\infty(D)$ and $L^\infty(\Omega; dP)$ respectively, and if

$$\sum_{m=1}^{\infty} \sqrt{\lambda_m} < \infty, \tag{5.6}$$

then the Karhunen-Loève (5.4) converges uniformly on $D \times \Omega$.

Below, $a(\boldsymbol{x}, \omega)$ is approximated by a *deterministic* function $a_M(\boldsymbol{x}, \omega)$ of the first M random variables $\{X_m\}_{m=1}^M$ by truncating the Karhunen-Loève expansion (5.4). Since truncation of the Karhunen-Loève expansion after M terms will later be seen to lead to an $M + d$ dimensional deterministic problem, the complexity of our approach strongly depends on the size of M which in turn (compare (5.6)) depends on the decay of the Karhunen-Loève Eigenvalues λ_m.

5.2.1 Decay Properties of the Karhunen-Loève Eigenvalues

Decay criteria for the Karhunen-Loève Eigenvalue sequence $\{\lambda_m\}_{m\geq 1}$ are crucial, since the Eigenvalue decay determines the complexity of the stochastic Galerkin FEM as we shall see below.

The decay rates for the Karhunen-Loève Eigenvalues are shown to depend on the regularity of the covariance kernel V_a. Roughly speaking, the smoother the covariance kernel of the coefficient, the faster the Karhunen-Loève Eigenvalues decay, with analyticity implying exponential decay and finite Sobolev regularity giving rise to algebraic decay. Remarkably, these results hold true already for *piecewise* regularity of the covariance kernel. For proofs, refer to [99, 107].

Definition 5.7 (Piecewise regularity) *Let D be a bounded domain of $\mathbb{R}^d$. A correlation function $V : D \times D \to \mathbb{R}$ is said to be* piecewise analytic/smooth/$H^{p,q}$ *on $D \times D$ if there exists a finite sequence $\{D_j\}_{j=1}^J$ of subdomains of D such that $\overline{D} = \bigcup_{j=1}^J \overline{D}_j$ and V is analytic/smooth/$H^p \otimes H^q$ in an open neighbourhood of $\overline{D}_j \times \overline{D}_{j'}$ for any pair (j, j').*

The decay rates of the Eigenvalues depend heavily on the regularity of the kernel V:

Proposition 5.8 (Eigenvalue decay rates) *Let $V \in L^2(D \times D)$, $D \subset \mathbb{R}^d$ be a symmetric correlation kernel defining a compact, self-adjoint and positive integral operator via*

$$\mathcal{V} : L^2(D) \to L^2(D), \quad (\mathcal{V}u)(\boldsymbol{x}) = \int_D V(\boldsymbol{x}, \boldsymbol{x}')u(\boldsymbol{x}')\, d\boldsymbol{x}'. \tag{5.9}$$

Let $\{\lambda_m\}_{m\geq 1}$ be the Eigenvalue sequence of $\mathcal{V}$.

- Analytic kernel:
  If V is piecewise analytic on $D \times D$, then there exist constants $c_1, c_2 > 0$ such that
  $$0 < \lambda_m \leq c_1 \exp\left(-c_2 m^{1/d}\right) \quad \forall m \geq 1. \tag{5.10}$$

- Entire kernel:
  Let V be a Gaussian covariance kernel given by
  $$V(\boldsymbol{x}, \boldsymbol{x}') := \sigma^2 \exp\left(-\frac{|\boldsymbol{x} - \boldsymbol{x}'|^2}{\gamma^2 \delta^2}\right) \tag{5.11}$$
  where $\sigma, \gamma > 0$ are real parameters and δ is the diameter of the domain D. Then, there holds
  $$0 < \lambda_m \lesssim \sigma^2 \frac{(1/\gamma)^{m^{1/d}+2}}{\Gamma(m/2^{1/d})} \quad \forall m \geq 1, \tag{5.12}$$
  where Γ is the Gamma function interpolating the factorial.

- Sobolev kernel:
  If $V \in L^2(D \times D)$ is symmetric and piecewise $H^{p,0}$ ($p \geq 1$), then
  $$0 < \lambda_m \lesssim m^{-(2p-1)/d} \quad \forall m \geq 1. \tag{5.13}$$

Remark 5.14

- *Many covariances which occur in engineering practice are piecewise analytic [92].*

- *The parameters σ and γ of the entire kernel in* (5.11) *are referred to as the* standard deviation *and the* correlation length *of a respectively (given V is the covariance of a as shown in Section 5.1). This kernel admits an analytic continuation to the whole complex space $\mathbb{C}^d$ (an* entire function*). Therefore, the Eigenvalue decay is even faster than in* (5.10).

- *The decay estimate* (5.12) *is sub-exponential in dimension $d > 1$. This is essentially due to the higher multiplicity of the Eigenvalues in dimensions larger than 1. To visualise this effect, the largest 2000 Eigenvalues of the three dimensional, factorisable kernel $V(\boldsymbol{x}, \boldsymbol{x}') = \exp(-10|\boldsymbol{x} - \boldsymbol{x}'|^2)$ are plotted in Figure 5.2 together with a theoretical estimate obtained by dropping the (asymptotically negligible) numerator $(1/\gamma)^{m^{1/d}+2}$ in* (5.12).

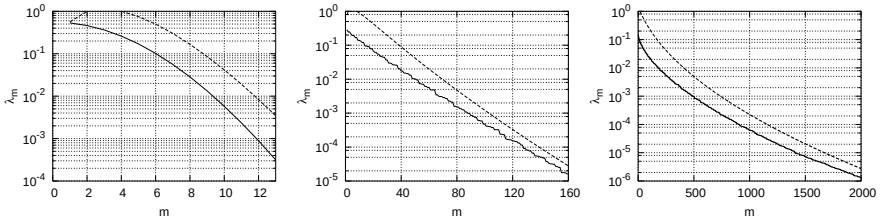


Figure 5.2: Eigenvalue decay of analytic covariance kernel $\exp(-10|\boldsymbol{x} - \boldsymbol{x}'|^2)$ in one (left plot), two (middle) and three dimensions (right). Plotted are the Eigenvalues λ_m against their index m (solid line), multiplicity taken into account. The dashed line shows the approximation $1/\Gamma(m/2^{1/d})$ from (5.12). The domain D is the unit box $(-1, 1)^d$.

- *The Eigenvalue decay rate of a Sobolev kernel can only deteriorate by a multiplicative factor in case of a small correlation length.*

Example 5.15

- *The kernel*

$$V(\boldsymbol{x}, \boldsymbol{x}') := \sigma^2 \exp\left(-\frac{|\boldsymbol{x} - \boldsymbol{x}'|^{1+\varepsilon}}{\gamma^{1+\varepsilon}\delta^{1+\varepsilon}}\right)$$

  admits only Sobolev regularity for $0 \leq \varepsilon < 1$. δ *is the diameter of* D.

- *For* $D = (-1, 1)$, *the first ten Eigenvalues of the analytic kernel* $V(x, x') = \exp(-|x - x'|^2)$ *are plotted in Figure 5.3. The plot also shows the first ten Eigenvalues of the kernel* $V(x, x') = \exp(-|x - x'|^{1+\varepsilon})$ *for various values of* ε.

5.2.2 Karhunen-Loève Eigenfunction Estimates

The point-wise convergence of the Karhunen-Loève expansion is essential for the error control in (5.2) when truncating the Karhunen-Loève expansion after M terms. So, beside criteria ensuring a fast decay of the Eigenvalues, the Eigenfunctions of the covariance kernel need to be estimated in the $L^\infty(D)$ norm. It turns out that the smoothness assumption on the covariance kernel allows also a good point-wise control of the Eigenfunctions.

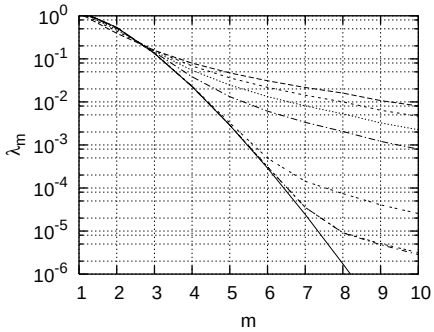


Figure 5.3: Eigenvalue decay depends on the regularity of the kernel. The plot is in one dimension, $D = (-1, 1)$ showing the Eigenvalues for the kernel $\exp(-|x - x'|^{1+\varepsilon})$ for (from top to bottom) $\varepsilon = 0$, $\varepsilon = 1/4$, $\varepsilon = 1/2$, $\varepsilon = 3/4$, $\varepsilon = 9/10$, $\varepsilon = 0.99$, $\varepsilon = 0.999$ (all have Sobolev regularity) and $\varepsilon = 1$ (an entire kernel, solid line).

Proposition 5.16 *Let $V \in L^2(D \times D)$ be symmetric and piecewise smooth. Denote by $\{\lambda_m, \varphi_m\}_{m\geq 1}$ the sequence of Eigenpairs of the associated covariance operator via* (5.9)*, such that $\|\varphi_m\|_{L^2(D)} = 1, \forall m \geq 1$. Then, for any $s > 0$ and any multi-index $\boldsymbol{\alpha} \in \mathbb{N}^d$, there holds*

$$\|\partial^{\boldsymbol{\alpha}} \varphi_m\|_{L^\infty(D)} \lesssim |\lambda_m|^{-s} \quad \forall m \geq 1,$$

where the constant depends on $s, \boldsymbol{\alpha}$ and J (Definition 5.7).

5.3 Stochastic Galerkin Method

In this section, the stochastic Galerkin method used to solve (5.2) is developed along with appropriate error estimates. In a first step, a is replaced by its Karhunen-Loève expansion a_M. Secondly, the random variables X_m in a_M are replaced by deterministic variables $y_m \in \mathbb{R}$. These y_m are discretised with a spectral Finite Element Method. Eventually, detailed algorithms sum up the whole process and show how (5.2) can be solved approximately in parallel.

Throughout this section, assume that the diffusion coefficient a satisfies (5.1) and possesses a Karhunen-Loève expansion (5.4) such that the following *assumptions* are satisfied:

1. The family $X = \{X_m\}_{m\geq 1}$ of random variables is uniformly bounded in the probability space $L^\infty(\Omega, dP)$, *i. e.*

$$\exists c_X > 0, \quad \|X_m\|_{L^\infty(\Omega,dP)} \leq c_X \in \mathbb{R} \quad \forall m \geq 1. \tag{5.17}$$

2. The family $X = \{X_m\}_{m\geq 1}$ of random variables is independent.

Remark 5.18 *The error analysis of* (5.2) *when truncating the Karhunen-Loève series* (5.4) *of a can be carried out without Assumption 2—it is made only to simplify the exposition (for details see [107]).*

5.3.1 Truncation of the Karhunen-Loève Expansion of a

Assumption 1, coupled with the decay estimates in Proposition 5.8, implies the uniform convergence on $D \times \Omega$ of the Karhunen-Loève expansion of a.

For any $M \in \mathbb{N}$, define the truncated coefficient

$$a_M(\boldsymbol{x}, \omega) = E_a(\boldsymbol{x}) + \sum_{m\geq 1}^{M} \sqrt{\lambda_m}\varphi_m(\boldsymbol{x})X_m(\omega). \tag{5.19}$$

The following point-wise error estimates for the truncated coefficient hold, depending on the smoothness of the coefficient $a(\boldsymbol{x}, \omega)$.

Proposition 5.20 (Truncated coefficient error estimates) *If V_a is piecewise analytic/smooth on $D \times D$ and* (5.17) *holds, then the Karhunen-Loève expansion of a converges uniformly on $D \times \Omega$ at the rate*

$$\|a - a_M\|_{L^\infty(D\times\Omega)} \lesssim \begin{cases} \exp\left(-c_2(1/2 - s)M^{1/d}\right) & V_a \text{ pw. analytic} \\ M^{1-(p-1)(1-2s)/d} & V_a \text{ pw. smooth} \end{cases} \quad \forall M \in \mathbb{N}, \tag{5.21}$$

for any $s > 0$, $p \geq 1$ and with a constant depending on the spatial dimension d, s from Proposition 5.16, c_1 and c_2 from Proposition 5.8, c_X from Assumption 1 and J from Definition 5.7.

Remark 5.22

- *Since p can be chosen arbitrarily large and s arbitrarily close to 0 in* (5.21)*, the remainder of the Karhunen-Loève series of a after truncation is rapidly and uniformly decaying on $D \times \Omega$.*

- *If the family $X = \{X_m\}_{m\geq 1}$ is independent (Assumption 2), it can be shown that the ellipticity (therefore the existence and uniqueness of a solution) of the stochastic problem* (5.2) *is preserved with the same upper and lower bounds α, β in* (5.1) *for any $M \geq 0$ when replacing the diffusion coefficient a by a_M.*

Combining Proposition 5.20 and a Strang-type argument shows that the error due to replacing the diffusion coefficient a by its truncated Karhunen-Loève expansion a_M in (5.2) is rapidly decaying as $M \to \infty$, at least in the case of piecewise analytic/smooth covariance kernel V_a. This is essential since the number of terms M retained in the Karhunen-Loève expansion will later determine the deterministic dimension necessary for the stochastic Galerkin method.

Proposition 5.23 *Consider a diffusion coefficient a satisfying* (5.1) *such that V_a is piecewise analytic/smooth on $D \times D$. If u and u_M are the solutions of* (5.2) *and*

$$-\operatorname{div}(a_M(\boldsymbol{x},\omega)\nabla_x u_M(\boldsymbol{x},\omega)) = f(\boldsymbol{x}) \tag{5.24}$$

respectively, then

$$\|u-u_M\|_{\mathcal{H}_0^1(D)} \lesssim \|u\|_{\mathcal{H}_0^1(D)} \cdot \begin{cases} \exp\left(-c_2(1/2-s)M^{1/d}\right) & \text{if } V_a \text{ piecewise analytic} \\ M^{1-(p-1)(1-2s)/d} & \text{if } V_a \text{ piecewise smooth,} \end{cases}$$

for all $M \geq 0$ and $p \geq 1$.

5.3.2 Associated Deterministic Problem

We study (5.24), obtained by truncation at level M of the Karhunen-Loève expansion of the diffusion coefficient a in (5.2). The random variables X_m in (5.19) are replaced by deterministic variables $y_m \in \mathbb{R}$. Without loss of generality, $c_X = 1/2$ (Assumption 1) is assumed in the following, so that for $\{X_m\}_{m\geq 1}$ in (5.4), Range $X_m \subset I := [-1/2, 1/2]$ $\forall m \geq 1$. Denote by ρ_m the probability measure associated to the random variable X_m,

$$\rho_m(\mathcal{B}) := P(X_m \in \mathcal{B}) \text{ for any Borel set } \mathcal{B} \subseteq \mathbb{R},$$

and define a probability measure on $\mathbb{R}^M$ by $\rho := \rho_1 \times \rho_2 \times \ldots \times \rho_M$.[2] Associate the mapping $\tilde{a}_M$ with a_M by

$$\begin{aligned} &\tilde{a}_M : D \times I^M \to \mathbb{R}, \\ &(\boldsymbol{x}, y_1, \ldots, y_M) \mapsto E_a(\boldsymbol{x}) + \sum_{m=1}^{M} \sqrt{\lambda_m}\varphi_m(\boldsymbol{x}) y_m. \end{aligned} \tag{5.25}$$

Consider the deterministic problem with the variational form
Find $\tilde{u}_M \in H_0^1(D) \otimes L^2(I^M, d\rho)$ such that

$$-\operatorname{div}(\tilde{a}_M(\boldsymbol{x}, \boldsymbol{y})\nabla_x \tilde{u}_M(\boldsymbol{x}, \boldsymbol{y})) = f(\boldsymbol{x}) \tag{5.26}$$

The uniform ellipticity of all truncates a_M (which follows from (5.21) for large M or Remark 5.22) ensures the well-posedness of (5.26).

The solution of (5.24) can be obtained by solving (5.26) by backward substitution:

Proposition 5.27 *If $\tilde{u}_M$ is the solution of* (5.26) *and u_M solves* (5.24)*, then*

$$u_M(\boldsymbol{x}, \omega) = \tilde{u}_M(\boldsymbol{x}, X_1(\omega), \ldots, X_M(\omega)),$$

$\lambda \times P$*-a. e.* $(\boldsymbol{x}, \omega) \in D \times \Omega$.

5.3.3 Stochastic Regularity

The solution $\tilde{u}_M$ of the deterministic elliptic problem (5.26) has an analytic extension into a subset of $\mathbb{C}^M$ as is shown below. As simplification, assume that $\rho \sim \lambda$ is the Lebesgue measure on I^M.

Remark 5.28 *This assumption is very strong: it implies that the random variables $\{X_m\}_{m\geq 1}^M$ are independently identically distributed with a uniform distribution on $[-1/2, 1/2]$.*

The solution $\tilde{u}_M$ solves
Find $\tilde{u}_M \in H_0^1(D) \otimes L^2(I^M)$ such that

$$-\operatorname{div}(\tilde{a}_M(\boldsymbol{x}, \boldsymbol{y})\nabla_x \tilde{u}_M(\boldsymbol{x}, \boldsymbol{y})) = f(\boldsymbol{x}). \tag{5.29}$$

[2] Note that $\rho \neq P$ (P ist the continuous probability measure in (5.1)–(5.3)): $\rho \to P$ as $M \to \infty$ 'in law'. However, $\rho(\Omega) = 1\ \forall M$, *i. e.* ρ is a probability measure on Ω.

If the number of terms M retained in the truncated Karhunen-Loève expansion is large, the number of degrees of freedom necessary for the accurate solution of Problem (5.29) appears to be prohibitive. However, this is not the case in general due to favourable regularity properties of the solution $\tilde{u}_M(\boldsymbol{x}, \boldsymbol{y})$ with respect to $\boldsymbol{y}$.

To show this, a result on stochastic regularity is presented which allows to show that the computational effort in solving (5.29) is moderate, even for large M.

Proposition 5.30 *Let $\tilde{u}_M$ be the solution of* (5.26)*. Then, as an $H_0^1(D)$-valued function on I^M, $\tilde{u}_M$ can be analytically extended to an open neighbourhood of I^M in $\mathbb{C}^M$, whose size in y_m is increasing for $m \to M$ like $1/\nu_m$, where $\nu_m := \sqrt{\lambda_m}\|\varphi_m\|_{L^\infty(D)}$ for $m \geq 1$.*

5.3.4 Stochastic Spectral Discretisation

The analyticity of $\tilde{u}_M$ as a function of $\boldsymbol{y}$ ensures an exponential convergence rate of its Finite Element approximations obtained by a p-method with respect to $\boldsymbol{y}$.

Therefore, define for $r \in \mathbb{N}$, the space of polynomials of degree at most r,

$$\mathcal{P}_r := \operatorname{span}\{1, t, t^2, \ldots, t^r\} \subset L^2(I)$$

and, for $\boldsymbol{r} = (r_1, r_2, \ldots, r_M) \in \mathbb{N}^M$, an anisotropic polynomial space by tensor product

$$\mathcal{P}_{\boldsymbol{r}} := \bigotimes_{i=1}^{M} \mathcal{P}_{r_i} \subset L^2(I^M).$$

Further, for $\boldsymbol{r} \in \mathbb{N}^M$, we shall denote by $\tilde{u}_{M,\boldsymbol{r}}$ the solution of the variational problem (5.29) in the subspace $H_0^1(D) \otimes \mathcal{P}_{\boldsymbol{r}}$,

$$\int_{I^M}\int_D \tilde{a}_M(\boldsymbol{x}, \boldsymbol{y})\nabla_x \tilde{u}_{M,\boldsymbol{r}}(\boldsymbol{x}, \boldsymbol{y}) \cdot \nabla_x v(\boldsymbol{x}, \boldsymbol{y})\, d\boldsymbol{x}\, d\boldsymbol{y} = \int_{I^M}\int_D f(\boldsymbol{x}) v(\boldsymbol{x}, \boldsymbol{y})\, d\boldsymbol{x}\, d\boldsymbol{y}. \quad \forall v \in H_0^1(D) \otimes \mathcal{P}_{\boldsymbol{r}}. \quad (5.31)$$

Based on the quasi-optimality of any Galerkin projection of (5.29) and on Proposition 5.30, the convergence rate of the $\boldsymbol{y}$-semi-discretisation of (5.29) is estimated in terms of the overall numberof deterministic problems $N_{\boldsymbol{r}}$ to be solved, *independently of the number of terms M retained in the truncated Karhunen-Loève expansion.*

Proposition 5.32 *Let $a \in L^\infty(D \times \Omega)$ satisfy* (5.1)*. Suppose that V_a is piecewise analytic such that* (5.10) *holds with two strictly positive constants c_1 and c_2. Define*

$$r_m := \left\lceil \frac{M^{1/d}}{m^{1/d}} \right\rceil \quad \forall 1 \leq m \leq M. \tag{5.33}$$

Then, there holds, with constants depending only on c_1 and c_2 and with c_3 a true constant,

$$N_{\boldsymbol{r}} := \dim \mathcal{P}_{\boldsymbol{r}} \lesssim e^{c_3 M}$$

and

$$\|\tilde{u}_M - \tilde{u}_{M,\boldsymbol{r}}\|_{H_0^1(D)\otimes L^2(I^M)} \lesssim \exp\left(-c_2 M^{1/d}\right) \lesssim \exp\left(-c_2 c_3^{-1/d} (\log N_{\boldsymbol{r}})^{1/d}\right). \tag{5.34}$$

Remark 5.35

- *Due to the $\boldsymbol{y}$-analyticity of $\tilde{u}_M$, one can show that the asymptotic error estimate* (5.34) *also holds in $H_0^1(D) \otimes L^\infty(I^M)$.*

- *The convergence rate* (5.34) *is algebraic for $d = 1$ and sub-algebraic for $d > 1$, which makes the computation in these cases rather expensive. However, using different polynomial FE spaces for the $\boldsymbol{y}$-discretisation, which are not of tensor product type, the convergence rate can be improved beyond algebraic. The construction of these spaces as well as their properties will be addressed in [107].*

  Nonetheless, the use of tensor product FE spaces has an important advantage: an appropriate choice of the basis decouples the problem into exactly $N_{\boldsymbol{r}}$ deterministic diffusion problems, which can be solved in parallel (see Section 5.3.6).

- *In Proposition 5.32, exact Eigenpairs $\{(\lambda_m, \varphi_m)\}_m$ of the Karhunen-Loève expansion are assumed. However, a similar result holds for approximated Eigenpairs $\{(\lambda_m^h, \varphi_m^h)\}_m$ after choosing $M \approx |\log h|^d$, in order to balance the Karhunen-Loève truncation error and the Eigenvalue discretisation error (c. f. [107]).*

1. Choose a steering parameter $0 < \theta \leq 1$.
2. Choose an overkill level $K \in \mathbb{N}$ (in practice $K \leq 15$).
3. Compute
$$\tilde{u}_{M,k\cdot \boldsymbol{e}_m} \quad \forall 1 \leq k \leq K, 1 \leq m \leq M,$$
with $\{\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots \mathbf{e}_M\}$ a basis of $\mathbb{R}^M$.
4. Compute the decay rate of the relative error (size of domain of analyticity of $\tilde{u}_M$ with respect to y_k)
$$\eta_{m,k} := \frac{\|\tilde{u}_{M,k\cdot \mathbf{e}_m} - \tilde{u}_{M,(k-1)\cdot \boldsymbol{e}_m}\|}{\|\tilde{u}_{M,(k-1)\cdot \boldsymbol{e}_m}\|}$$
for all $1 \leq k \leq K, 1 \leq m \leq M$.
5. Initialise the polynomial degree $\boldsymbol{r} := (0, 0, \ldots, 0) \in \mathbb{N}^M$.
6. Compute the 'active' stochastic dimensions
$$\mathcal{M}_\theta := \left\{ m : \eta_{m,r_m+1} \geq \theta \cdot \max_{1\leq n\leq M} \eta_{n,r_n+1} \right\}.$$
7. Compute the new polynomial degree (raise r_m for all $m \in \mathcal{M}_\theta$)
$$\boldsymbol{r}_{\text{new}} := \boldsymbol{r} + \sum_{m\in,\mathcal{M}_\theta} \boldsymbol{e}_m$$
8. If $\max_m r_m < K$ goto 6 otherwise stop.

Algorithm 5.1: Adaptive selection of stochastic degree r_m.

5.3.5 Adaptive Selection of Stochastic Degree

Proposition 5.32 gave an error estimate of the spectral discretisation in the stochastic variable based on the assumption of piecewise analyticity of the correlation function $V_a(\boldsymbol{x}, \boldsymbol{x}')$ in $D \times D$ and based on the a-priori selection (5.33) of the stochastic polynomial degrees r_m which are in turn based on the stochastic regularity result Proposition 5.30.

Alternatively, it is possible to numerically determine the polynomial degree $\boldsymbol{r}$, using Algorithm 5.1 which successively identifies the coordinates y_m in which the largest change in the Finite Element solution occurs when the polynomial degree r_m is increased.

Remark 5.36 (Simplification of Algorithm 5.1)

- *Algorithm 5.1 to generate the adaptive polynomial degree* $\boldsymbol{r}$ *can be simplified by replacing the stochastic PDE by a stochastic algebraic equation,*

$$\Big(\beta_0 + \sum_{m=1}^{M} \beta_m y_m\Big) u = 1$$

where

$$\beta_0 := \inf_D E_a \text{ and } \beta_m := \sqrt{\lambda_m}\|\varphi_m\|_{L^\infty(D)} \simeq \sqrt{\lambda_m} \text{ for } m \geq 1.$$

- *Moreover,* $\{\eta_{m,k}\}_{k=1}^{K}$ *can be computed only for a small value of* K *and use a-priori knowledge (exponential decay in* k*) to predict* $\eta_{m,k}$ *for all* $k > K$ *by linear regression on* $\{\log \eta_{m,k}\}_{k=1}^{K}$.

Figures 5.4–5.7 show the results obtained using the algebraic version of Algorithm 5.1 (Remark 5.36) for different correlation lengths ($\gamma = 1,\ 1/2,\ 1/5$ and $1/10$) on the unit square and the L-shaped domain. The plots all have the same scales for the Eigenvalues λ_m (shown on the left ordinate) and the polynomial degree r_m (shown on the right ordinate).

5.3.6 Complete Algorithm

The analyticity of the covariance kernel V_a can be exploited to semi-discretise (5.31) with respect to $\boldsymbol{y}$ at a polynomial convergence rate, as stated in Proposition 5.32.

It is possible to obtain the solution of the semi-discrete problem (5.31) numerically by solving a large number (depending on $\boldsymbol{r}$) of *independent* deterministic elliptic boundary value problems with different data: the corresponding algorithm is derived below. Consequently, to compute the solution of (5.31), already available deterministic solvers combined with the algorithm derived in the following can be used.

The semi-discretisation of (5.31) with respect to $\boldsymbol{y}$ can be done using any basis of $\mathcal{P}_{\boldsymbol{r}}$. Generally, this results in a coupled system of deterministic elliptic boundary value problems. However, there exists a choice of a basis of $\mathcal{P}_{\boldsymbol{r}}$ which leads to a

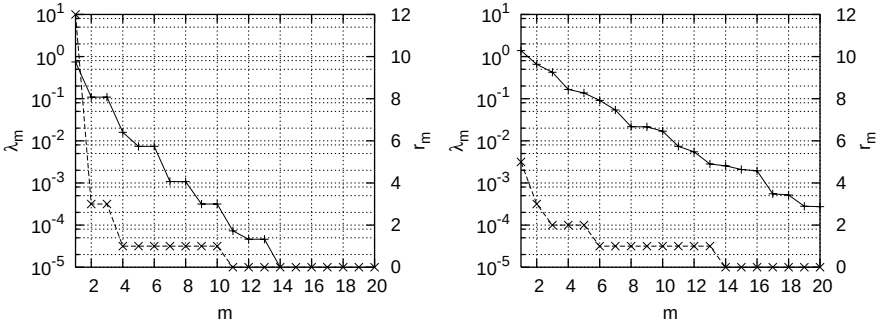


Figure 5.4: Eigenvalue decay (left ordinate, solid line) and adaptive polynomial degree (obtained using the algebraic version of the Algorithm 5.1) (right ordinate, dashed line) in the unit square (left plot) and L-shaped domain (right plot) for the entire kernel $\exp(-|\boldsymbol{x}-\boldsymbol{x}'|^2)$ (correlation length $\gamma = 1$) in two dimensions. The polynomial degree $\boldsymbol{r}$ results in 26,624 and 165,888 deterministic problems on the left and right respectively.

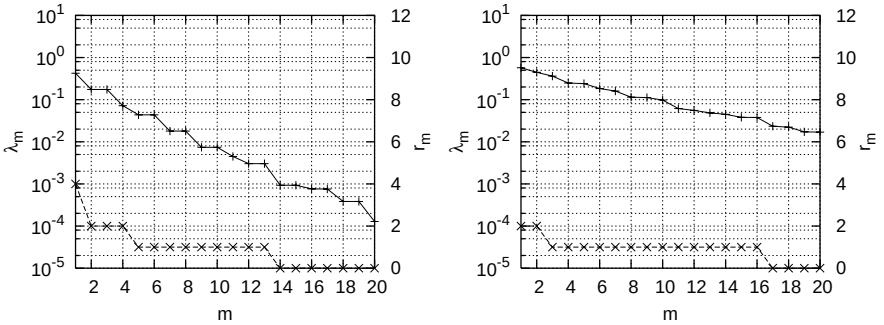


Figure 5.5: Eigenvalue decay (left ordinate, solid line) and adaptive polynomial degree (obtained using the algebraic version of the Algorithm 5.1) (right ordinate, dashed line) in the unit square (left plot) and L-shaped domain (right plot) for the entire kernel $\exp(-4|\boldsymbol{x}-\boldsymbol{x}'|^2)$ (correlation length $\gamma = 1/2$) in two dimensions. The polynomial degree $\boldsymbol{r}$ results in 69,120 and 147,456 deterministic problems on the left and right respectively.

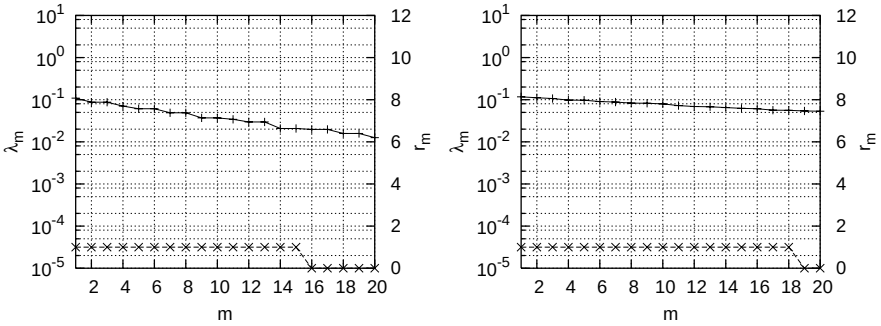


Figure 5.6: Eigenvalue decay (left ordinate, solid line) and adaptive polynomial degree (obtained using the algebraic version of the Algorithm 5.1) (right ordinate, dashed line) in the unit square (left plot) and L-shaped domain (right plot) for the entire kernel $\exp(-25|\boldsymbol{x} - \boldsymbol{x}'|^2)$ (correlation length $\gamma = 1/5$) in two dimensions. The polynomial degree $\boldsymbol{r}$ results in 32,768 and 262,144 deterministic problems on the left and right respectively.

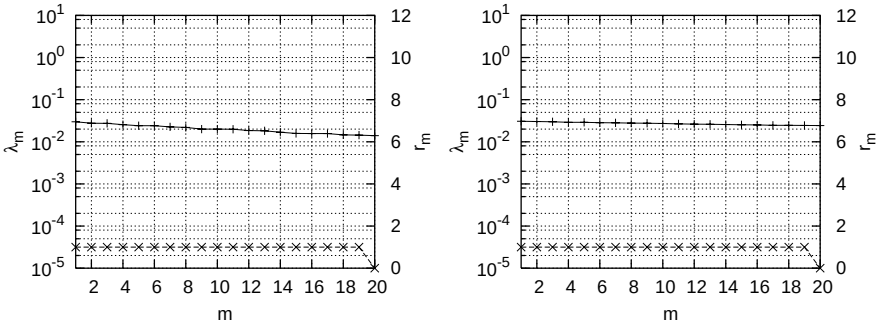


Figure 5.7: Eigenvalue decay (left ordinate, solid line) and adaptive polynomial degree (obtained using the algebraic version of the Algorithm 5.1) (right ordinate, dashed line) in the unit square (left plot) and L-shaped domain (right plot) for the entire kernel $\exp(-100|\boldsymbol{x} - \boldsymbol{x}'|^2)$ (correlation length $\gamma = 1/10$) in two dimensions. The polynomial degree $\boldsymbol{r}$ results in 524,288 and 524,288 deterministic problems on the left and right respectively.

decoupled system. Denote, for $r \in \mathbb{N}$, the Eigenpairs of the symmetric bilinear form

$$(u, v) \mapsto \int_{-1/2}^{1/2} u(t)v(t)t\,dt \tag{5.37}$$

in $\mathcal{P}_r := \operatorname{span}\{1, t, t^2, \ldots, t^r\}$ by $\{\mu_{j,r}, P_{j,r}\}_{j=0}^{r}$.

For convenience of notation, define on the index set $\mathbb{N}^M$ the ordering

$$\boldsymbol{j} \leq \boldsymbol{r} \iff 0 \leq j_m \leq r_m \quad \forall 1 \leq m \leq M.$$

Further, set

$$P_{\boldsymbol{j},\boldsymbol{r}} := \bigotimes_{i=1}^{M} P_{j_i,r_i} \tag{5.38}$$

for $\boldsymbol{j} \leq \boldsymbol{r} \in \mathbb{N}^M$. Clearly,

$$\mathcal{P}_{\boldsymbol{r}} = \operatorname{span}\{P_{\boldsymbol{j},\boldsymbol{r}} : 0 \leq j_m \leq r_m \forall 1 \leq m \leq M\},$$

and $\{P_{\boldsymbol{j},\boldsymbol{r}}\}_{\boldsymbol{j}\leq\boldsymbol{r}}$ is the basis of $\mathcal{P}_{\boldsymbol{r}}$ used to decouple the semi-discrete problem.

Proposition 5.39 *For a given $\boldsymbol{r} \in \mathbb{N}^M$, let $\tilde{u}_{M,\boldsymbol{r}}$ be the solution of* (5.31)*. For every multi-index $\boldsymbol{j} \leq \boldsymbol{r}$, denote by $\tilde{u}_{M,\boldsymbol{j}} \in H_0^1(D)$ the solution of the deterministic diffusion problem in D*

$$-\operatorname{div}(\tilde{a}_{M,\boldsymbol{j}} \nabla \tilde{u}_{M,\boldsymbol{j}}) = f_{\boldsymbol{j}} \tag{5.40}$$

where

$$\begin{aligned} \tilde{a}_{M,\boldsymbol{j}}(\boldsymbol{x}) &:= E_a(\boldsymbol{x}) + \sum_{m=1}^{M} \sqrt{\lambda_m} \cdot \varphi_m(\boldsymbol{x}) \mu_{j_m,r_m}, \\ f_{\boldsymbol{j}}(\boldsymbol{x}) &:= f(\boldsymbol{x}) \cdot \prod_{m=1}^{M} \int_{-1/2}^{1/2} P_{j_m,r_m}(t)\,dt. \end{aligned} \tag{5.41}$$

Then,

$$\tilde{u}_{M,\boldsymbol{r}}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{\boldsymbol{j}\leq\boldsymbol{r}} \tilde{u}_{M,\boldsymbol{j}}(\boldsymbol{x}) P_{\boldsymbol{j},\boldsymbol{r}}(\boldsymbol{y}). \tag{5.42}$$

The statistics of u_M, solution to (5.24), can be then obtained by backward substitution, via Proposition 5.27. For the simplest statistics, the mean and the correlation, there holds

Proposition 5.43 *If u_M solves* (5.24) *and $\tilde{u}_{M,\boldsymbol{j}}$ solves* (5.40) *for all $\boldsymbol{j} \leq \boldsymbol{r}$, then*

$$E_{u_M}(\boldsymbol{x}) = \sum_{\boldsymbol{j} \leq \boldsymbol{r}} \tilde{u}_{M,\boldsymbol{j}}(\boldsymbol{x}) \prod_{m=1}^{M} \int_{-1/2}^{1/2} P_{j_m,r_m}(y_m)\, dy_m$$

$$C_{u_M}(\boldsymbol{x}, \boldsymbol{x}') = \sum_{\boldsymbol{j} \leq \boldsymbol{r}} \tilde{u}_{M,\boldsymbol{j}}(\boldsymbol{x}) \tilde{u}_{M,\boldsymbol{j}}(\boldsymbol{x}') \prod_{m=1}^{M} \int_{-1/2}^{1/2} P_{j_m,r_m}(y_m)^2\, dy_m$$

Algorithm 5.2 summarises the steps developed for solving (5.2).

5.4 Fast Computation of Karhunen-Loève Expansion

In order to use the (truncated) Karhunen-Loève expansion (5.4) in practice, its first M Eigenpairs must be computed efficiently and accurately in arbitrary domains D. In one dimension, for particular kernels, explicit Eigenfunctions are known (*e. g.* [60]). These can be used to obtain explicit Eigenpairs also for multidimensional tensor product domains D, if $V_a(\boldsymbol{x}, \boldsymbol{x}')$ is separable. This is often the case in subsurface flow problems, where D is a box.

To deal with random coefficients in arbitrary geometries, however, an efficient numerical approximation of the Eigenpairs of the operator associated to the covariance kernel via (5.9) is an essential step in solving (5.2). Note that only the Eigenpairs $\{\lambda_m, u_m\}$ with $\lambda_m \neq 0$ are of interest. The Karhunen-Loève Eigenvalue problem reads in variational form:
Find λ_m and $0 \neq \varphi_m \in L^2(D)$ such that

$$\int_{D\times D} V_a(\boldsymbol{x}, \boldsymbol{x}')\varphi_m(\boldsymbol{x}')v(\boldsymbol{x})\, d\boldsymbol{x}'\, d\boldsymbol{x} = \lambda_m \int_D \varphi_m(\boldsymbol{x})v(\boldsymbol{x})\, d\boldsymbol{x} \quad \forall v \in L^2(D). \tag{5.47}$$

Since the Eigenpairs of V_a are used to approximate the diffusion coefficient a, L^∞ approximations of the Eigenfunctions are needed.

1. *Computation of the deterministic part of the Karhunen-Loève expansion of a:*
   - Assume V_a.
   - Choose truncation order M.
   - Compute the first M Eigenpairs $\{\lambda_m, \varphi_m\}_{m=1}^{M}$ of V_a. (5.44)
2. *Computation of the polynomial basis used for semi-discretisation:*
   - Compute anisotropic polynomial degree $\boldsymbol{r} = (r_m)_{m=1}^{M} \in \mathbb{N}^M$ according to Algorithm 5.1.
   - Compute Eigenpairs $\{\mu_{j,r_m}, P_{j,r_m}\}_{j=0}^{r_m}$ in $\mathcal{P}_{r_m} := \operatorname{span}\{1, t, t^2, \ldots, t^{r_m}\}$ of (5.45)
   $$(u, v) \to \int_{-1/2}^{1/2} u(t)v(t)t\,dt.$$
3. *Semi-discretisation:*
   - Assume E_a and f.
   - Compute $\tilde{u}_{M,\boldsymbol{j}}$ solution of the deterministic diffusion problem in D (5.46)
   $$-\operatorname{div}(\tilde{a}_{M,\boldsymbol{j}}\nabla\tilde{u}_{M,\boldsymbol{j}}) = f_{\boldsymbol{j}}$$
   where
   $$\tilde{a}_{M,\boldsymbol{j}}(x) = E_a(\boldsymbol{x}) + \sum_{m=1}^{M} \sqrt{\lambda_m}\cdot\varphi_m(\boldsymbol{x})\mu_{j_m,r_m},$$
   $$f_{\boldsymbol{j}}(\boldsymbol{x}) = f(\boldsymbol{x})\cdot\prod_{m=1}^{M}\int_{-1/2}^{1/2} P_{j_m,r_m}(t)\,dt$$
   and for all $\boldsymbol{j} = (j_1, j_2, \ldots, j_M) \in \mathbb{N}^M$ with $0 \le j_m \le r_m \quad \forall 1 \le m \le M$.
4. *Post-processing:*
   - Assume $\{X_m\}_{m=1}^{M}$.
   - Compute statistics of $u_M(\boldsymbol{x}, \omega)$ via backward substitution
   $$u_M(\boldsymbol{x}, \omega) = \sum_{\boldsymbol{j}} \tilde{u}_{M,\boldsymbol{j}}(\boldsymbol{x})\cdot\prod_{m=1}^{M} P_{j_m,r_m}(X_m(\omega))$$

Algorithm 5.2: Complete algorithm for solving (5.2).

To compute Karhunen-Loève Eigenpairs, a FE discretisation of (5.47) with piecewise constants on a regular triangulation $\mathcal{T}_h$ of D with mesh-width h is used (which will later also be used for the FE approximation of (5.2)). Assume that $\mathcal{T}_0$ is partitioned as in Definition 5.7. This ensures that a is analytic/smooth/$H^p \otimes H^q$ in an open neighbourhood of every element of $\mathcal{T}_0$, depending on the (piecewise) regularity of a in D. Let $\mathcal{T}_h$ be an extension of $\mathcal{T}_0$. Let

$$\mathcal{S}^{p,0}(\mathcal{T}_h, D) := \{v \in L^2(D) : v|_K \in \mathcal{P}_p \quad \forall K \in \mathcal{T}_h\}$$

denote the FE space of discontinuous, piecewise polynomials of order p on $\mathcal{T}_h$.

Then, the Galerkin approximation of (5.47) with the Finite Element space $V_N = \mathcal{S}^{p,0}(\mathcal{T}_h, D) \subset L^2(D)$ reads:
Find $0 \neq \lambda_m^h, \varphi_m^h \in V_N$ such that

$$\int_{D\times D} V_a(\boldsymbol{x}, \boldsymbol{x}')\varphi_m^h(\boldsymbol{x}')v(\boldsymbol{x})\,d\boldsymbol{x}'\,d\boldsymbol{x} = \lambda_m^h \int_D \varphi_m^h(\boldsymbol{x})v(\boldsymbol{x})\,d\boldsymbol{x} \quad \forall v \in V_N. \quad (5.48)$$

Proposition 5.49 *Suppose that $a \in L^\infty(D \times \Omega)$ such that $V_a \in H^{p+1}(D) \otimes L^2(D)$. Let (λ_m, u_m) be an Eigenpair of V_a with $\lambda_m \neq 0$. Then, for every $p \geq 0$ holds for $h \to 0$*

$$\|\varphi_m - \varphi_m^h\|_{L^\infty(D)} \lesssim h^{p+1}, \qquad |\lambda_m - \lambda_m^h| \lesssim h^{p+1}$$

where the constants depend on V_a and m.

For the proof, refer to [99].

The calculation of Karhunen-Loève Eigenpairs involves the solution of the dense matrix Eigenproblem corresponding to (5.48), *i. e.* of

$$\underline{V}\boldsymbol{\varphi} = \lambda \underline{M}\boldsymbol{\varphi}. \quad (5.50)$$

Here, both matrices $\underline{V}$ and $\underline{M}$ are symmetric and positive definite, with $\underline{M}$ being diagonal if the basis of $\mathcal{S}^{0,0}(\mathcal{T}_h, D)$ is chosen as the characteristic functions of the elements $K \in \mathcal{T}_h$.

For physical domains D in three dimensions and realistic meshes $\mathcal{T}_h$, the size of the Eigenproblem can be as large as 10^6 and standard Eigensolvers are not applicable.

An iterative Eigensolver based on Krylov subspaces [58, 59] which requires only matrix vector multiplies is used. The multiplication $\boldsymbol{\varphi} \mapsto \underline{V}\boldsymbol{\varphi}$ is done in

$O(N \log N)$ operations using a variant of the Fast Multipole Method for general kernels.

The main idea of the Fast Multipole Method for general kernels is to expand the kernel $V(\boldsymbol{x}, \boldsymbol{x}')$ in a series which decouples the variables $\boldsymbol{x}$ and $\boldsymbol{x}'$. Truncation of the expansion introduces a controllable error for $\boldsymbol{x}$ and $\boldsymbol{x}'$ far away from each other. On the other hand, evaluating the truncated expansion of the kernel and especially $\boldsymbol{\varphi} \mapsto \underline{V}^f \boldsymbol{\varphi}$ is very fast (where $\underline{V}^f$ denotes the matrix for the far field only). The near field matrix $\underline{V}^n$ for $\boldsymbol{x}$ and $\boldsymbol{x}'$ close together is then evaluated traditionally resulting in a sparse matrix.

The following subsections review the Fast Multipole Method for general kernels [93, 94].

5.4.1 Kernel Expansions in Fast Multipole Methods for General Kernels

Definition 5.51 (Valid kernel expansion) *Let $\eta \in [0, 1)$ and $\mathcal{I}$ an index set and $V : D \times D \to \mathbb{C}$ a kernel function. Then, V is said to have a* valid kernel expansion *if for all $\boldsymbol{x}_0, \boldsymbol{x}_0' \in D$, $\boldsymbol{x}_0 \neq \boldsymbol{x}_0'$ and expansion orders $m \in \mathbb{N}$, there exists an approximation V_m of the form*

$$\begin{aligned} V(\boldsymbol{x}, \boldsymbol{x}') &\approx V_m(\boldsymbol{x}, \boldsymbol{x}'; \boldsymbol{x}_0, \boldsymbol{x}_0') \\ &:= \sum_{(\mu,\nu)\in\mathcal{I}_m} \varkappa_{(\mu,\nu)} X_\mu(\boldsymbol{x}; \boldsymbol{x}_0) Y_\nu(\boldsymbol{x}'; \boldsymbol{x}_0') \end{aligned}$$

for $\mathcal{I}_m \subset \mathcal{I} \times \mathcal{I}$ such that for all $\boldsymbol{x}, \boldsymbol{x}' \in D$ satisfying

$$|\boldsymbol{x} - \boldsymbol{x}_0| + |\boldsymbol{x}' - \boldsymbol{x}_0'| \leq \eta |\boldsymbol{x}_0 - \boldsymbol{x}_0'|$$

the error is bounded by

$$|V(\boldsymbol{x}, \boldsymbol{x}') - V_m(\boldsymbol{x}, \boldsymbol{x}'; \boldsymbol{x}_0, \boldsymbol{x}_0')| \leq C \exp(-C(\eta) m) \, |\boldsymbol{x} - \boldsymbol{x}'|^{-\hat{s}} \tag{5.52}$$

with $C(\eta) > 0$ a decreasing function and C a constant both independent of m. $\hat{s}$ denotes the singularity order of the kernel for $\boldsymbol{x} = \boldsymbol{x}'$.

Čebyšev Kernel Expansion

V is interpolated by Čebyšev polynomials. The Čebyšev polynomial of the first kind on $I = [-1, 1]$ for $\mu \in \mathbb{Z}$ reads

$$T_\mu(x) = \cos(\mu \arccos(x)).$$

The Čebyšev interpolant of f defined on I is given by

$$f_m(x) := \sum_{\substack{\mu\in\mathbb{Z}\\ |\mu|<m}} \hat{f}_\mu T_\mu(x), \text{ where } \hat{f}_\mu := 1/m \sum_{i=0}^{m} f(x_i) T_\mu(x_i)$$

for the Čebyšev points $x_i = \cos((i + 1/2)\pi/m) \in I$, *i. e.* the m roots of T_m. In higher dimensions, a tensor product Ansatz is used.

Definition 5.51 is verified for kernels of the form $V(x, x') = V(x' - x)$ admitting an analytic extension into $\mathbb{C}^d \setminus \{0\}$ in [93, 94]. The Čebyšev kernel expansion reads

$$V_m(x, x'; x_0, x_0') = \sum_{\substack{(\mu,\nu)\in\mathbb{N}_0^d\times\mathbb{N}_0^d\\ |\mu+\nu|_\infty<m}} \frac{(x_0 - x)^\mu}{\mu!} \frac{(x' - x_0')^\nu}{\nu!} \cdot (\mu + \nu)! c_{\mu+\nu}(x_0, x_0'), \tag{5.53}$$

where the coefficients c_i, $i \in \mathbb{N}_0^d$, are defined by the polynomial interpolation

$$\sum_{\substack{\mu\in\mathbb{Z}^d\\ |\mu|_\infty<m}} \hat{f}_\mu(x_0, x_0') T_\mu\left(\frac{z}{\eta\|x_0 - x_0'\|_\infty}\right) = \sum_{\mu\in\mathbb{N}_0^d, |\mu|_\infty<m} c_\mu(x_0, x_0') z^\mu, \tag{5.54}$$

$$f(\xi; x_0, x_0') := V\big(\chi(\xi; x_0' - x_0)\big),$$
$$\chi(\xi; x_0' - x_0) := \eta \left\|x_0' - x_0\right\| \cdot \xi + x_0' - x_0.$$

Remark 5.55 *The evaluation of the coefficients $\hat{f}_\mu$ in* (5.54) *requires $\mathcal{O}(m^d)$ evaluations of V at the Čebyšev points of order m. However, the kernel V is not explicitly needed (in closed form) but can be given by a subroutine. It is trivial to adapt the Čebyšev kernel expansion to new kernels. Therefore, it suites our needs very well.*

Kernels V which are only piecewise analytic in $D \times D$ with $\overline{D} = \bigcup_{j=1}^{J} \overline{D}_j$ should be considered separately in each $D_j \times D_{j'}$ (*c. f.* Definition 5.7). There, the requested analytic extension is possible.

Other Kernel Expansions

Other kernel expansions which are primarily used in boundary element methods are presented in [93, 94]: Taylor expansion [65, 64], Multipole expansion [16] and fast Helmholtz solvers [34, 35].

5.4.2 Cluster Expansion

The previous section showed how a kernel can be approximated. In general, this approximation is not valid for all $(\boldsymbol{x}, \boldsymbol{x}') \in D \times D$. In order to define a *global approximation* on $D \times D$, a collection of local approximations is used, where each of the local approximations is associated with an appropriate block of a given partition of $D \times D$. Those blocks are called *clusters* and the combination of local approximations a *cluster expansion.*

More precisely, denote by $\mathcal{P}(D)$ the set of all subsets of D.

Definition 5.56 (Čebyšev radius and centre) *For any set $A \subset \mathbb{R}^d$, the Čebyšev ball of A is the smallest ball containing A, i. e. define*

- *the* Čebyšev radius $\check{r}_A$ *by*

$$\check{r}_A := \inf_{\boldsymbol{x} \in \mathbb{R}^d} \sup_{\boldsymbol{x}' \in A} |\boldsymbol{x}' - \boldsymbol{x}|.$$

- *the* Čebyšev centre $\check{\boldsymbol{c}}_A$ *by*

$$\check{r}_A = \sup_{\boldsymbol{x}' \in A} |\boldsymbol{x}' - \check{\boldsymbol{c}}_A|.$$

Definition 5.57 (Far and near field) *Suppose $\mathcal{C} \subset \mathcal{P}(D) \times \mathcal{P}(D)$ to be a finite partition of $D \times D$ and let $\eta \in (0, 1)$. An element $(\sigma, \tau) \in \mathcal{C}$ is called* η-cluster *iff*

$$\check{r}_\sigma + \check{r}_\tau \leq \eta |\check{\boldsymbol{c}}_\sigma - \check{\boldsymbol{c}}_\tau|.$$

The set of all η-clusters in $\mathcal{C}$,

$$\mathcal{F} := \mathcal{F}(\mathcal{C}, \eta) = \{(\sigma, \tau) \in \mathcal{C} : (\sigma, \tau) \text{ is an } \eta\text{-cluster}\}$$

is called the far field *of grain η and its complement $\mathcal{N} := \mathcal{N}(\mathcal{C}, \eta) = \mathcal{C} \setminus \mathcal{F}(\mathcal{C}, \eta)$ the associated* near field.

Definition 5.58 (Cluster expansion) *Let the kernel $V(\boldsymbol{x}, \boldsymbol{x}')$ satisfy the Definition 5.51. Then, for an analytic kernel V,*

$$V_m(\boldsymbol{x}, \boldsymbol{x}') := \begin{cases} V_m(\boldsymbol{x}, \boldsymbol{x}'; \check{\boldsymbol{c}}_\sigma, \check{\boldsymbol{c}}_\tau) & \text{if } (\boldsymbol{x}, \boldsymbol{x}') \in \sigma \times \tau \text{ and } (\sigma, \tau) \in \mathcal{F}, \\ V(\boldsymbol{x}, \boldsymbol{x}') & \text{otherwise} \end{cases}$$

for all $(\boldsymbol{x}, \boldsymbol{x}') \in D \times D, \boldsymbol{x} \neq \boldsymbol{x}'$ is a so-called cluster expansion *of the kernel $V(\boldsymbol{x}, \boldsymbol{x}')$. For a piecewise analytic kernel V, define*

$$V_m(\boldsymbol{x}, \boldsymbol{x}') := \begin{cases} V_m^{jj'}(\boldsymbol{x}, \boldsymbol{x}'; \check{\boldsymbol{c}}_\sigma, \check{\boldsymbol{c}}_\tau) & \text{if } (\boldsymbol{x}, \boldsymbol{x}') \in \sigma \times \tau, (\sigma, \tau) \in \mathcal{F}^{jj'}, \\ & \qquad \sigma \subset D_j \text{ and } \tau \subset D_{j'}, \\ V(\boldsymbol{x}, \boldsymbol{x}') & \text{otherwise}, \end{cases}$$

where $\mathcal{F}^{jj'}$ is the far field of $D_j \times D_{j'}$.

Proposition 5.59 (Local error bound) *By construction, the local error bound in* (5.52) *remains valid for a cluster expansion:*

$$|V(\boldsymbol{x}, \boldsymbol{x}') - V_m(\boldsymbol{x}, \boldsymbol{x}')| \leq C_0 (C_1 \eta)^m |V(\boldsymbol{x}, \boldsymbol{x}')|$$

for all $(\boldsymbol{x}, \boldsymbol{x}') \in D \times D, \boldsymbol{x} \neq \boldsymbol{x}'$.

Hence, the matrix vector multiplication $\boldsymbol{\varphi} \mapsto \underline{V}\boldsymbol{\varphi}$ is approximated by

$$\boldsymbol{\varphi} \mapsto \tilde{\underline{V}}\varphi := \underline{V}^n \boldsymbol{\varphi} + \underline{V}^f \boldsymbol{\varphi}, \tag{5.60}$$

where $\underline{V}^n$ is the near field matrix

$$\left[\underline{V}^n\right]_{i,j} := \int_{\substack{\sigma \times \tau \\ (\sigma,\tau) \in \mathcal{N}}} V_a(\boldsymbol{x}, \boldsymbol{x}') \varphi_j(\boldsymbol{x}') \varphi_i(\boldsymbol{x}) \, d\boldsymbol{x}' \, d\boldsymbol{x},$$

φ_i are basis functions of $\mathcal{S}^{p,0}(\mathcal{T}_h, D)$, and $\underline{V}^f$ is the far field approximation by

$$\boldsymbol{\varphi} \mapsto \underline{V}^f \boldsymbol{\varphi} := \sum_{j,j'=1}^{J} \sum_{(\sigma,\tau)\in\mathcal{F}^{jj'}} \underline{X}_\sigma^\top \left(\underline{F}_{\sigma\tau}^{jj'} (\underline{Y}_\tau \boldsymbol{\varphi}) \right). \tag{5.61}$$

The matrices $\underline{X}_\sigma$, $\underline{Y}_\tau$ and $\underline{F}_{\sigma\tau}$ are defined by

$$\begin{aligned} \left[\underline{X}_\sigma\right]_{\boldsymbol{\mu},j} &:= \int_\sigma X_{\boldsymbol{\mu}}(\boldsymbol{x}; \check{\boldsymbol{c}}_\sigma)\varphi_j(\boldsymbol{x})d\boldsymbol{x} \\ \left[\underline{Y}_\tau\right]_{\boldsymbol{\nu},i} &:= \int_\tau Y_{\boldsymbol{\nu}}(\boldsymbol{x}'; \check{\boldsymbol{c}}_\tau)\varphi_i(\boldsymbol{x}')d\boldsymbol{x}' \\ \left[\underline{F}_{\sigma\tau}^{jj'}\right]_{\boldsymbol{\mu},\boldsymbol{\nu}} &:= \varkappa_{(\boldsymbol{\mu},\boldsymbol{\nu})}^{jj'}(\check{\boldsymbol{c}}_\sigma, \check{\boldsymbol{c}}_\tau) \end{aligned}$$

for $(\boldsymbol{\mu}, \boldsymbol{\nu}) \in \mathcal{I}_m$.

Remark 5.62

- *The cluster algorithm is only based on Definition 5.51. This ensures the exponential convergence with respect to the expansion order m. In addition, it leads to a low rank approximation of the far field part of $\underline{V}$.*
- *In the acceleration of the matrix vector multiplication $\boldsymbol{\varphi} \mapsto \underline{V}\boldsymbol{\varphi}$ it is essential that the matrices $\underline{X}_\sigma$, $\underline{Y}_\tau$ and $\underline{F}_{\sigma\tau}$ are never formed explicitly. Typically, the entries $[\underline{F}_{\sigma,\tau}]_{\boldsymbol{\mu},\boldsymbol{\nu}}$ only depend on $\boldsymbol{\mu} + \boldsymbol{\nu}$ with $|\boldsymbol{\mu} + \boldsymbol{\nu}| < m$. Therefore, only $\mathcal{O}(m^p)$ instead of $\mathcal{O}(m^{2p})$ entries have to be evaluated and stored ($p \in \{2, 3\}$ depends on the chosen kernel expansion).*
- *The Čebyšev kernel expansion* (5.53) *preserves the symmetry of the kernel V: $V_m(\boldsymbol{x}, \boldsymbol{x}') = V_m(\boldsymbol{x}', \boldsymbol{x})$. If, in addition, the given partition $\mathcal{C}$ is symmetric, i. e. $(\sigma, \tau) \in \mathcal{C} \Rightarrow (\tau, \sigma) \in \mathcal{C}$, then, $\underline{\tilde{V}}$ in* (5.60) *is also symmetric for Galerkin discretisations.*

5.4.3 Cluster Algorithm

The main goal is to efficiently (in terms of storage and computation time) realise (5.61) in the matrix vector multiplication (5.60). To this end, an appropriate partition $\mathcal{C}$ is needed.

- If $|A| < c$, return $(\{A\}, \emptyset)$,
- else
  - $(A_0, A_1) := \mathsf{split}(A)$.
  - $(\mathcal{V}_i, \mathcal{E}_i) := \mathsf{tree}(A_i, c)$ for $i = 0, 1$.
  - Return $(\mathcal{V}_0 \cup \mathcal{V}_1 \cup \{A\}, \mathcal{E}_0 \cup \mathcal{E}_1 \cup \{(A, A_0), (A, A_1)\})$.

Algorithm 5.3: $(\mathcal{V}, \mathcal{E}) = \mathsf{tree}(A, c)$. Generates a tree $(\mathcal{V}, \mathcal{E})$ out of a set A of cells of the domain D. $\mathcal{V}$ is the set of vertices and $\mathcal{E}$ the set of edges in the tree. Split bisects a set A into two disjoint sets A_0 and A_1 such that the Čebyšev radius of both sets is reduced.

An efficient way is to use a hierarchical decomposition of the mesh $\mathcal{T}$. Algorithm 5.3 generates such a decomposition, a so-called cluster tree:

Definition 5.63 (Cluster tree) *A* cluster tree $\mathcal{B}(A)$ *of a finite set A consists of subsets of* $\mathcal{P}(A) \setminus \emptyset$ *where*

- $\mathrm{root}(\mathcal{B}(A)) := A \in \mathcal{B}(A)$,
- $\sigma_i \cap \sigma_j \in \{\emptyset, \sigma_i, \sigma_j\}$ *for* $\sigma_i, \sigma_j \in \mathcal{B}(A)$

holds. The subsets $\sigma \subset \mathcal{B}(A)$ *are called* nodes *of the tree* $\mathcal{B}(A)$ *or* clusters *of A. The* children *of a node* σ *are the subsets* $\sigma_i \in \mathcal{B}(A)$, $i \in \mathcal{I}_\sigma$ *with* $|\mathcal{I}_\sigma|$ *minimal, satisfying*

$$\sigma_i \subset \sigma, \qquad \sigma_i \neq \sigma, \qquad \sigma = \bigcup_{i \in \mathcal{I}_\sigma} \sigma_i .$$

A node σ *of* $\mathcal{B}(A)$ *is called a* leaf *if it has no children, i. e.* $\mathcal{I}_\sigma = \emptyset$.

For a piecewise analytic kernel V, a cluster tree of each D_j in Definition 5.7 has to be computed leading to a multi-rooted cluster tree.

The hierarchical decomposition of the mesh is used in Algorithm 5.4 to generate the partition $\mathcal{C}$ of the mesh into its far field $\mathcal{F}$ and its near field $\mathcal{N}$ by calling $\mathsf{partition}(\mathcal{T}, \mathcal{T})$. This partition is symmetric in the sense of Remark 5.62. For a piecewise analytic kernel V, the far fields $\{\mathcal{F}^{jj'}\}_{j,j'=1}^{J}$ have to be computed by J^2 calls to partition.

- If
$$\Big(\bigcup_{a\in A} a, \bigcup_{b\in B} b\Big)$$
is an η-cluster, then return $(\emptyset, \{(A, B)\})$,
- else:
  - Let $A' := \mathsf{children}(A)$ and $B' := \mathsf{children}(B')$. (5.64)
  - Return
$$\begin{cases} \bigcup_{\substack{a\in A',\\ b\in B'}} \mathsf{partition}(a, b) & \text{if } A' \neq \emptyset \text{ and } B' \neq \emptyset \text{ and } |A| = |B|, \\ \bigcup_{a\in A'} \mathsf{partition}(a, B) & \text{if } A' \neq \emptyset \text{ and } (|A| > |B| \text{ or } B' = \emptyset), \\ \bigcup_{b\in B'} \mathsf{partition}(A, b) & \text{if } B' \neq \emptyset \text{ and } (|B| > |A| \text{ or } A' = \emptyset), \\ (\{(A, B)\}, \emptyset) & \text{otherwise}. \end{cases}$$

Algorithm 5.4: $(\mathcal{N}, \mathcal{F}) = \mathsf{partition}(A, B)$. The tree generated with tree in Algorithm 5.3 serves as mean to define the children of a set of cells in (5.64).

The matrix vector multiplication $\boldsymbol{\varphi} \mapsto \underline{V}\boldsymbol{\varphi} =: \boldsymbol{v}$ in (5.60) is evaluated in five steps:

1. Compute the near field contribution $\boldsymbol{\varphi}^n := \underline{V}^n\boldsymbol{\varphi}$.
2. Compute $\boldsymbol{u}_\tau := \underline{Y}_\tau\boldsymbol{\varphi}$ for all τ.
3. Compute
$$\boldsymbol{u}_\sigma := \sum_{(\sigma,\tau)\in\bigcup_{j,j'}\mathcal{F}^{jj'}} \underline{F}^{jj'}_{\sigma\tau}\boldsymbol{u}_\tau.$$
4. Compute
$$\boldsymbol{\varphi}^f := \sum_\sigma \underline{X}_\sigma^\top \boldsymbol{u}_\sigma.$$
5. Sum up the near field and far field contributions $\boldsymbol{v} := \boldsymbol{\varphi}^n + \boldsymbol{\varphi}^f$.

The steps 2–5 compute the far field contribution given in (5.61). The steps 2 and 4 can be accelerated by once more taking advantage of the hierarchical decomposition. The idea is to not store the matrices $\underline{X}_\sigma$ and $\underline{Y}_\tau$ but represent them using the hierarchical decomposition and so-called shift operators. For details, we refer to [93, 94].

5.4.4 Overall Error and Complexity

The following result [93, 94] shows the overall error caused by the discretisation and the approximation of $\underline{V}$ by $\tilde{\underline{V}}$. The result is first given for a source problem: Given $f \in H^{-s/2}(D)$, find $u \in H^{s/2}(D)$ such that

$$(\mathcal{V}(u), v) = a(u, v) = (f, v) \quad \forall v \in H^{s/2}(D), \tag{5.65}$$

where $\mathcal{V}(u)$ is given by (5.9). (5.65) is solved with $V_N = \mathcal{S}^{p,p}(D,)$ for $p = 0, 1$.

Proposition 5.66 *Assume that the bilinear form in* (5.65) *is coercive,* $s < 2p + 1$ *and* $t > 0$*. Let the cluster approximation* V_m *of the far field satisfy Definition 5.51 with a sufficiently small grain* η*. Choose the expansion order* m *according to*

$$m \leq C|\log h|$$

with $C > 0$ *sufficiently large.*

- *There exists* $h_0 > 0$ *such that the Galerkin discretisation with the perturbed matrix* $\tilde{\underline{V}}$ *in* (5.60) *is stable.*
- *The Galerkin solution of* $\tilde{\underline{V}}\boldsymbol{u} = \boldsymbol{f}$ *converges with*

$$\|u - u_N\|_{H^{s/2}(D)} \leq Ch^{\min(t,\, p+1-s/2)} \, \|f\|_{H^{-s/2+t}(D)} \,. \tag{5.67}$$

According to Proposition 1.37, the convergence (5.67) also holds for Eigenproblems such as (5.47)—the Eigenvalues converge twice as fast as the Eigenvectors.

The complexity of the algorithm is essentially log-linear [93]:

Proposition 5.68 (Complexity of the cluster algorithm) *Let* J^2 *be the number of analyticity domains in Definition 5.7. Then, the complexity of the cluster algorithm is* $\mathcal{O}(N \log^5 N J^2)$ *and the storage requirements are* $\mathcal{O}(N \log^4 N J^2)$.

```
blocksize=1000       ♯ number of deterministic FE problems in a block
nofproblems=7776          ♯ total number of deterministic FE problems

start=0
end=0
while [ $start -lt $nofproblems ]; do          ♯ loop over all blocks
    let "end = start + blocksize - 1"
    if [ $end -ge $nofproblems ]; then
        let "end = nofproblems - 1"
    fi
    ♯ submit job to queueing system
    qsub -v start=$start,end=$end -l nodes=1 diffusion_job
    let "start += $blocksize"
done
```

Algorithm 5.5: Master script (Bash syntax)

5.5 Parallel Solution of Deterministic Problems

As the deterministic Finite Element problems (Proposition 5.40) are all completely independent, they can be solved in parallel on a Beowulf type cluster [18]. This parallelisation is achieved by a simple shell script (referenced as the *master script*) which runs on the administration node of a Beowulf cluster. These deterministic FE problems are so-called “embarrassingly parallel”.

The *master script* (Algorithm 5.5) is given the total number of deterministic problems to be solved and the size of the blocks in which the deterministic problems should be grouped. Using this data, the master script sets up a job for every block of problems[3] in the queueing system of the Beowulf cluster [7] using a *job script*. The queueing system calls the job script when enough CPUs on the cluster are available. As we only request one CPU per job, this should happen fairly often. It is up to the queueing system to parallelise the whole process. If enough CPUs are available, all blocks are solved in parallel. If the cluster is nearly full (only very few CPUs available at the same time), it might even occur, that the whole problem is solved serially—this is the worst case scenario.

[3] A typical size of a block should give a run time of an hour or less, depending on the total number of problems.

```
mkdir -p $LOCAL_SCR
TEMPDIR=`mktemp -d $LOCAL_SCR/diffusion_XXXXXX`  # temporary directory

rcp $WORK_SRV:$PBS_O_WORKDIR/diffusion.tar.gz $TEMPDIR   # get archive
cd $TEMPDIR
tar xzf diffusion.tar.gz                          # unpack input archive

diffusion -f diffusion.concepts -s $start -e $end    # call FE solver

res_archive=results_s_${start}_e_${end}.tar.gz
tar czvf $res_archive alpha*vector diffusion.out    # results archive
rcp $res_archive $WORK_SRV:$PBS_O_WORKDIR          # send results back

cd $HOME
rm -rf $TEMPDIR
```

Algorithm 5.6: Job script (Bash syntax)

The *job script* (Algorithm 5.6) is aware of the archive `diffusion.tar.gz` containing the input data and the Finite Element solver and the indices of the problems via input arguments. This archive is then unpacked and the FE solver is run. One block of problems is solved by executing the FE solver once. It reads in the mesh and additional data and computes orthogonal polynomial basis (5.45) at start-up. Then, all the deterministic problems are solved serially one after another. According to (5.41), the stiffness matrix and the load vector have to be computed separately for every problem. Every deterministic FE solve generates one vector of coefficients. The resulting data of all deterministic problems in the block is again archived and sent back to the administration node of the cluster.

When all jobs generated by the master script have been run, a result archive from every block exists on the administration node. The collected data can then be used to do some post-processing. Most of the post-processing can again be done serially: computing the approximation of the mean and correlation of the quantity of interest u is not much more than a sum over all solutions of the deterministic problems (Proposition 5.43). More complicated statistics like probabilistic level sets

$$D_\varepsilon^\delta := \{\boldsymbol{x} \in D : P(|u(\boldsymbol{x}, .)| > \delta) < \varepsilon\}$$

might require additional Monte Carlo evaluation of stochastic integrals—which again can be performed in parallel.

5.6 Numerical Results

5.6.1 Software

Both key computational tasks from Algorithm 5.2 (computation of the Karhunen-Loève expansion and the solution of the deterministic problems) are done with the *same class library* Concepts (*c. f.* [26, 54, 93] and Part III). The main advantage is that transfering the data for the Karhunen-Loève expansion from (5.44) to (5.46) in Algorithm 5.2 is particularly easy due to shared data structures. The two different computational tasks are performed with two different main programs and on different hardware. More precisely, the computations are done on the same mesh $\mathcal{T}$[4] and the Fast Multipole solver which computes the Karhunen-Loève expansion assigns the values of the Eigenfunctions $\{\varphi_m\}$ to the cells of the mesh (which have a unique number). Therefore, the Finite Element solver is able to read the data and assign it to the correct cells using the same unique cell numbers. This makes it even possible to use a further refined mesh (compared to the Karhunen-Loève expansion) for the Finite Element computations.

Karhunen-Loève Expansion

The *Karhunen-Loève expansion* (5.44) in Algorithm 5.2 is solved by a generalised Fast Multipole Method (*c. f.* Section 5.4). In our present implementation, this step is performed serially since the complexity of the Fast Multipole Method is log-linear in N. We are able to treat reasonably large Finite Element meshes with several hundred thousand degrees of freedom with this serial implementation. As Ansatz functions, piecewise constants $\mathcal{S}^{0,0}(D, \mathcal{T})$ or piecewise linears $\mathcal{S}^{1,0}(D, \mathcal{T})$ are used. The Eigenproblem is solved using JDBSYM by Roman Geus and Oscar Chinellato [58, 59].

[4] Note that this assumption was made only for convenience of implementation—if an adaptive FE solver for the equation (5.40) is available, then for each $\boldsymbol{j}$, a different mesh adapted to the coefficient $\tilde{a}_{M,\boldsymbol{j}}$ could be created. This, however, would require more sophisticated post-processing when computing mean and variances of the stochastic Galerkin solution.

Deterministic Finite Element Problems

For the *deterministic Finite Element problems* (5.46), we use a linear or quadratic FEM (*i. e.* $\mathcal{S}^{1,1}(D, \mathcal{T})$ and $\mathcal{S}^{2,1}(D, \mathcal{T})$ respectively). The resulting linear system is solved using a a diagonally preconditioned conjugate gradient algorithm.

Orthogonal Polynomials

The computation of the *orthogonal polynomials* in step (5.45) is also important but not very complex. It amounts to solving matrix Eigenvalue problems of the type $\underline{A}\boldsymbol{x} = \lambda \underline{B}\boldsymbol{x}$ with symmetric and positive semi-definite matrices $\underline{A}$ and $\underline{B}$ of size r_j. Due to the product form (5.38) of the shape functions in the stochastic variable, only the univariate form (5.37) needs to be discretised. Since in the adaptive algorithms for the selection of the stochastic polynomial degrees the optimal degrees are a-priori unknown, the univariate generalised Eigenproblem needs to be solved for all possible polynomial degrees r_j which could occur. This could be done in MATLAB for degrees between 1 and 20 once and the Eigenpairs stored on disk for the main calculation. However, as these Eigenproblems are very moderate in size, we actually compute them during the start-up phase of a block of deterministic problems using LAPACK [75] (*c. f.* Section 5.5).

Post-processing

The post-processing is again done serially (mostly on the administrative node of the Beowulf cluster). We compute the expected value and variance of $\tilde{\boldsymbol{u}}_N$ and their respective L^2- and H^1-norms according to Proposition 5.43.

As in Section 5.3.3, we assume that the random variables $\{X_m\}_{m\geq 1}^{M}$ in (5.19) are independently identically distributed with a uniform distribution on $[-1/2, 1/2]$.

5.6.2 Hardware

The computations were performed on the following machines.

- The Karhunen-Loève expansion (*i. e.* the solution of the large Eigenproblem with generalised Fast Multipole Methods) was done on a Sun Fire 880 with 32 GBytes of physical main memory and 8 processors at 800 MHz—only using one processor, though. GCC 3 [57] was used to compile the software.

M	$\boldsymbol{r}$	P	M	$\boldsymbol{r}$	P
1	(0)	1	3	(5, 1, 1)	24
1	(1)	2	4	(5, 1, 1, 1)	48
1	(2)	3	6	(6, 1, 1, 1, 1, 1)	224
3	(3, 1, 1)	16	6	(7, 2, 2, 1, 1, 1)	576
3	(4, 1, 1)	20	8	(8, 2, 2, 1, 1, 1, 1, 1)	2592

Table 5.1: Adapted polynomial degrees in the stochastic variables for different truncation parameters M of the Karhunen-Loève expansion. These values are found with Algorithm5.1 ($\theta = 0.7$).

- The deterministic Finite Element problems were solved on the Linux Beowulf cluster Asgard [7]. The compute nodes have two Pentium III (Katmai) at 500 MHz and 1 GByte of main memory. The compute nodes are interconnected with fast Ethernet. Again, we used GCC 3 [57] to compile the software.

5.6.3 Computations

The model problem used is (5.2) on $D = (0, 1)^2$ with

$$E_a(\boldsymbol{x}) = 2 + x, \qquad V_a(\boldsymbol{x}, \boldsymbol{x}') = \exp(-|\boldsymbol{x} - \boldsymbol{x}'|^2), \qquad f(\boldsymbol{x}) = 1.$$

There is no exact solution available.

The Eigenvalue problem (5.48) was solved using $2 \cdot 4^4 = 512$ cells with piecewise constants: $V_N = \mathcal{S}^{0,0}_{\partial D}(\mathcal{T}_h, D)$ without using a Fast Multipole Method (but with the full matrix). The results of (5.48) of the chosen kernel $V_a(\boldsymbol{x}, \boldsymbol{x}')$ are depicted in Figure 5.4.

The deterministic FE problems were solved using $2 \cdot 4^{\text{level}}$ cells with piecewise linear or quadratic basis functions: $V_N = \mathcal{S}^{p,1}_{\partial D}(\mathcal{T}_h, D)$, where $p = 1$ or 2, with a diagonally preconditioned conjugate gradient solver.

Stochastic Galerkin Method

The plots in Figure 5.8 show the convergence history of the relative error of the H^1-norm squared of the expected value and the variance of the solution of the

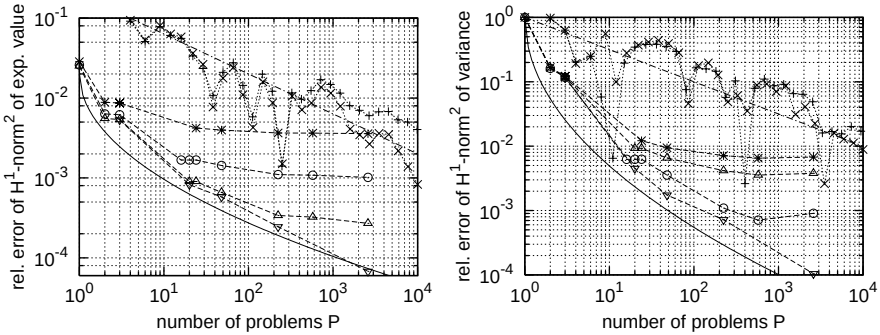


Figure 5.8: Relative error of the H^1-norms squared of the expected value (left) and the variance (right) plotted versus the number of problems P. The dotted lines show the Monte Carlo method and the dashed lines the stochastic Galerkin method. The dash-dotted lines show $\mathcal{O}(1/P^{1/2})$ and the solid lines give the sub-algebraic theoretical convergence (5.34).
Two different Monte Carlo computations are shown: both with $M = 15$, $+$ with linear FEM on refinement level 5 (961 degrees of freedom in the spatial discretisation) and $\times$ with quadratic FEM on level 8 (261121 degrees of freedom). The seeds of the pseudo-random number generator were the same for both computations.
Four different stochastic Galerkin computations with the adaptive degrees from Table5.1 are shown: $*$ with linear FEM on level 5 (961 dof), $\circ$ with linear FEM on level 6 (3969 dof), $\triangle$ with quadratic FEM on level 5 (3969 dof), $\triangledown$ with quadratic FEM on level 8 (261121 dof).

model problem. The plotted quantity in the case of the expected value is

$$\frac{\left\| E_{u_M}(\boldsymbol{x}) \right\|^2 - \overline{\left\| E_{u(\boldsymbol{x})} \right\|^2}}{\overline{\left\| E_{u(\boldsymbol{x})} \right\|^2}},$$

where $\overline{\|E_{u(\boldsymbol{x})}\|^2}$ was obtained by Rhomberg extrapolation of the last seven values of the adaptive stochastic degree computations with $M = 3, \ldots, 8$ and quadratic FEM on level 8. The same procedure was applied for the variance.

The convergence histories of the computations with different spatial resolutions in Figure 5.8 show that it is important to equilibrate the errors from different sources. In this case (ignoring the modelling error), error contributions come from the discretisation in the stochastic and spatial variables and the discretisation of the

Karhunen-Loève expansion (5.48). There are different parameters to control these discretisations: M and $\boldsymbol{r}$ for the stochastic variables and the polynomial degree p and the refinement level for the spatial variables. The Karhunen-Loève expansion (5.48) is controlled by the level only. The discretisation error of the Karhunen-Loève expansion (5.48) does not play a role in the range of parameters discussed here: We compared the convergence histories of two computations where only the level of the Karhunen-Loève expansion in (5.48) differed and they were identical. The same decrease of the convergence rate as seen for the coarser discretisations ($*$, $\circ$ and $\triangle$ in Figure 5.8) would also happen with the finest discretisation ($\triangledown$ in Figure 5.8 with quadratic FEM on level 8) if the number of problems P was higher: On these curves, only the stochastic resolution is increased, and the errors are not equilibrated.

Monte Carlo Method

Most of the code for the computations with the stochastic Galerkin method and the Monte Carlo method are identical. Only the diffusion coefficient, the right hand side and the post-processing differ slightly. The same methods for distributing the work load on a Beowulf type cluster as described in Section 5.5 can be applied to the Monte Carlo method.[5]

The Monte Carlo computations are done using (5.19) with a new realisation of $\{X_m\}_{m=1}^M$ for each $a_M(\boldsymbol{x}, \omega)$. The X_m are modelled as independently identically distributed random variables with a uniform distribution on $[-1/2, 1/2]$ (*c.f.* Remark 5.28). Each realisation y_m of X_m is computed using

$$y_m = \frac{\mathsf{rand()}}{\mathsf{RAND\_MAX}} - 1/2,$$

where rand() calls the pseudo-random number generator of the GNU C library returning a pseudo-random integer between 0 and RAND_MAX. The right hand side can be kept unchanged for all FE problems to be solved.

In the post-processing step, the expected value and the variance are computed using

$$E_{u_M}(\boldsymbol{x}) = \frac{1}{P}\sum_{i=1}^{P} \tilde{u}_i(\boldsymbol{x}), \qquad var_{u_M}(\boldsymbol{x}) = \frac{1}{P-1}\sum_{i=1}^{P} \left(\tilde{u}_i(\boldsymbol{x}) - E_{u_M}(\boldsymbol{x})\right)^2,$$

[5] It has to be taken care of properly seeding the pseudo-random number generator.

where P is the number of realisations of $a_M(\boldsymbol{x}, \omega)$ and $\tilde{u}_i(\boldsymbol{x})$ the resulting solution. The results taken at intermediate steps yield the convergence history shown in Figure 5.8. Apparently, the error of the H^1-norms squared of the expected value and the variance converge with $\mathcal{O}(1/P^{1/2})$.

Conclusions

The convergence histories of the stochastic Galerkin method (dashed lines) in Figure 5.8 indicate superior convergence properties compared to the Monte Carlo method (dotted lines):

- The convergence rate of the Monte Carlo method $\mathcal{O}(P^{1/2})$ is algebraic.
- Although the stochastic Galerkin method converges only with a sub-algebraic rate in two dimensions (5.34), it is able to beat the Monte Carlo method in our experiments.
- The stochastic Galerkin method performs better than the Monte Carlo method already in the pre-asymptotic range.
- Refining the spatial discretisation dramatically improves the approximation using the stochastic Galerkin method. On the other hand, the Monte Carlo method benefits from better spatial discretisation only by decreasing the convergence rate later (*c. f.* range $P = 10^3$ to 10^4 in Figure 5.8).
- The convergence of the stochastic Galerkin method is more predictable than the convergence of the Monte Carlo method.
- The discretisation in the stochastic variables can be controlled much better in the case of the stochastic Galerkin method using a non-uniform $\boldsymbol{r}$. The stochastic discretisation of the Monte Carlo method is only controlled by the truncation level M in (5.19).

Part III

Software: Concepts

6 hp-Finite Element Methods in Concepts

In the introduction of this thesis, we have defined our mission: Bijectivley map well-defined mathematical formulations of physical models to simulation software. We concentrate on variational formulations of operator equations given in the form Find $u \in U$ such that

$$a(u, v) = l(v) \quad \forall v \in V. \tag{6.1}$$

This problem and specific instances of it are mapped onto a hierarchy of classes implemented in C++ in the software Concepts [26, 54, 73, 74]. A short presentation of Concepts has already been given in the introduction to this thesis, it is introduced further in the first section of this chapter. In addition, the choice of C++ as an object oriented programming language shall be justified.

In the rest of the chapter, the main ingredients and their implementation of our hp-discretisation of $H^1_{\Gamma_D}(D)$ are explained: meshes and subdivisions, shape functions and global basis functions and fast integration techniques for the element matrices.

6.1 Introduction

6.1.1 Object Oriented Programming

C++ is a flexible, object oriented language supporting the high level paradigms *polymorphism (inheritance)* and *parametrised types (templates)*. On the other hand, it is still possible to write hand-optimised low level code to get maximal performance in critical parts of the code. Additionally, compilers with good optimisation capabilities are widely available [57]. C++ is widely used in both scientific and commercial applications.

The main advantage of object oriented programming (as opposed to structured programming as in the C and Fortran world) is the high level of abstraction which is possible. If a sufficiently high level of abstraction (comparable to the abstract mathematical formulation (6.1)) is reached a *unified and timeless design of smulation software is possible.* To reach such a high level of abstraction with a structured programming language would be a much more complex task.

Cleverly adding some low level code in critical parts to the otherwise high level software makes it possible to write C++ code which even outperforms Fortran [101].

UML Diagrams

The design of object oriented software can be described using UML diagrams (the Unified Modelling Language UML [89] is a graphical language). We use UML diagrams to describe the statics and the dynamics of Concepts.

A box in a UML diagram denotes a class (if the name is underlined, it denotes an object, *i. e.* an instance of a class). Italic identifiers are abstract (*i. e.* have no implementation behind), whereas underlined members are static (*i. e.* class and not object attributes). Arrows with empty heads (△) denote relations in a class hierarchy. Dashed arrows in a class hierarchy are used for specialisations of interfaces whereas solid arrows mean normal specialisation. An empty or filled diamond (◇ and ◆) at an arrow tail means aggregation: the class on the arrow head is tightly linked to the class on the arrow tail.

In the text, abstract classes are typeset in italic sans-serif like *AbstractClass* and concrete classes in sans-serif like NormalClass.

6.1.2 Basic Classes in Concepts

An application in Concepts typically performs the following steps:

1. Build a meshed domain of interest D. A mesh $\mathcal{T}$ is built from cells which contain the element maps.
2. Build a space V_N on the mesh. The space creates the elements. Typical for FEM and BEM: the elements are associated to cells in the mesh.[1]

[1] As introduced in Chapter 2, the *cells* are geometric entities (including coordinates and neighbourhood information). The *elements* add Finite Element information to the cells: polynomial degree, shape functions etc. (at least in the cases considered here).

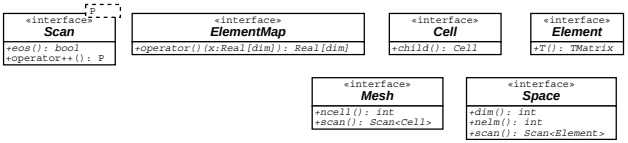


Figure 6.1: Classes in Concepts representing mesh, space and elements.

3. Build the system matrix $\underline{A}$ and the system vector $\boldsymbol{l}$ from the element contributions computed by the bilinear and linear forms $a(.,.)$ and $l(.)$ respectively.

4. Solve the resulting linear system with the solver.

Refer to the introduction (including Figure 0.1 on page 6) of this thesis for an example.

In Concepts, there are classes for the space, the mesh and the elements: see Figure 6.1. The main member of the classes *Mesh* and *Space* is *scan()*. In both classes, it returns a *Scan*<*P*> with the template parameter P set accordingly. The instance of *Scan* is a scanner over the space or the mesh and makes it possible to loop over the elements of the space or the cells of the mesh. Typically, the constructor of a space loops over all cells of the mesh and creates an associated element to every cell.[2] The most important member of the class *Element* is *T()*: It returns the T matrix (*c. f.* Section 3.3) of an element. The T matrix is used to assemble the local shape functions defined on the element into global basis functions of the space defined on the whole domain D. The element map $F_K : \hat{K} \rightarrow K$ is realized by *ElementMap*.

Figure 6.2 shows the classes of Concepts for the bilinear and linear form, the stiffness matrix and load vector and the linear solver. The constructor of the stiffness matrix SparseMatrix takes as arguments a space and a bilinear form. It assembles the stiffness matrix by looping over the elements of the space and calling the application operator operator() of the bilinear form on the elements. The same is done in the constructor of the load vector Vector. By using these abstract class declarations, it is possible to explicitly implement the assembly operator (*c. f.* Definition 3.22): Algorithm 6.1 shows the constructor of Vector, *i. e.* the assembling of

[2] However, there are methods where elements do not have a cell of the mesh associated to it. For this reason, *Element* in Figure 6.1 does not have a relation with *Cell*.

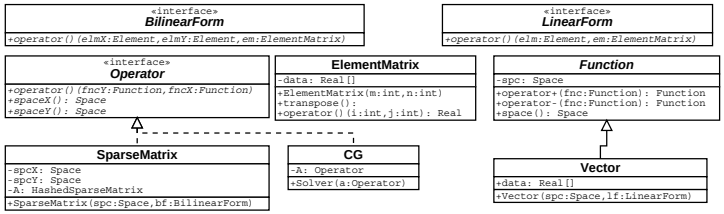


Figure 6.2: Classes in Concepts for bilinear and linear forms, stiffness matrix and load vector and the solver.

the global load vector—the assembly operator of a stiffness matrix looks similar. The solver is, like SparseMatrix, derived from the general class *Operator*. *Operator* is the realization of the general concept of an operator between vector spaces. A solver (in this case conjugate gradients CG) fits into this concept, too.

Remark 6.2 *The solver and matrix classes are either our own implementation (*CG, GMRes, SparseMatrix *and* DenseMatrix*) or interfaces to other packages (*SuperLU *[45, 46],* PETSc *[11, 12, 13],* Pardiso *[91] and* Umfpack *[38, 39, 40]).*

6.2 Mesh Classes in Concepts

This section is devoted to the classes in Concepts which handle the mesh $\mathcal{T}$. The first part explains the data structures in some detail. The second part shows how the different subdivision strategies for quadrilaterals and hexahedra are implemented.

6.2.1 Classes to Handle a Mesh

The most important part of a mesh is a scanner over all cells in the mesh (*c. f.* Section 6.1.2 and Figure 6.1). For one, two and three dimensions, there are separate specialisations of *Mesh*, namely *Mesh1*, *Mesh2* and *Mesh3* respectively. A realization of a specific mesh (like a line in one dimension) is then a specialisation of the respective interface class as shown in Figure 6.3.

The scanner of a mesh is used to loop over the cells of a mesh. A *Cell* consists of a topological entity of the respective dimension (for instance Edge, Triangle or

```
Vector::Vector(const Space& spc, LinearForm& lf) : v_(new F[n_]) {
  memset(v_, 0, n_ * sizeof(v_[0]));  // set the load vector to 0

  ElementMatrix A(3, 1), B(3, 1);     // initialize 2 element matrices

  Space::Scan* sc = space().scan();   // get a scanner to loop over the space
  while (*sc) {                       // loop over all elements of the space
    Element& elm = (*sc)++;           // get the current element
    const TMatrixBase& T = elm.T();   // get the T matrix of the element
    lf(elm, A);                       // evaluate the linear form
    A.transpose(); T(A, B);           // ** apply the T matrix **
    B.transpose();

    for(int i = T.n(); i--;) {        // add the element's contribution
      v_[T.index(i)] += B(i,0);       // into the global load vector
    }
  }
  delete sc;
}
```

Algorithm 6.1: Constructor of the class Vector which implements the assembly operator for a load vector (3.24). This does not depend on any particular implementation of the Finite Element space or the linear form but only on the abstract classes. The key point is the application of the T matrix: $(\boldsymbol{I}_K^\top \underline{T}_K)^\top = \underline{T}_K^\top \boldsymbol{I}_K$, *c. f.* (3.24).

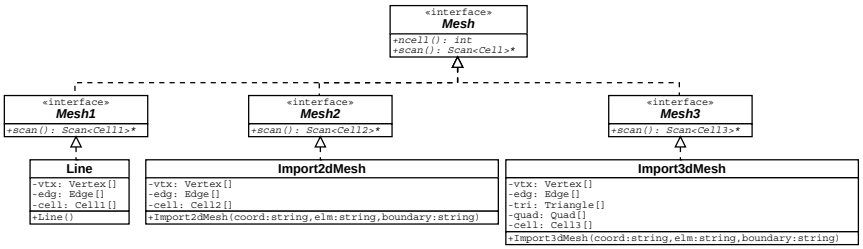


Figure 6.3: Classes for meshes I. An implementation of a concrete mesh is a sibling of Line, Import2dMesh or Import3dMesh depending on the spatial dimension. Import2dMesh and Import3dMesh can be used to import mesh descriptions from files (with the same structure as in [6]).

- Initialise an (empty) list of auxiliary data for edges.
- Loop over all cells in the mesh
  - For each edge of the cell
    - If the edge is not yet in the list, add it.
    - Register the current cell to the edge.

Algorithm 6.2: Finding neighbours over edges in a two dimensional mesh. This algorithm creates a temporary list where all edges are stored including links to their adjacent cells.

Hexahedron) and an element map. For each of these topological entities, there is a specialisation of *Cell* (this is not shown). The cell links topological and geometrical information.

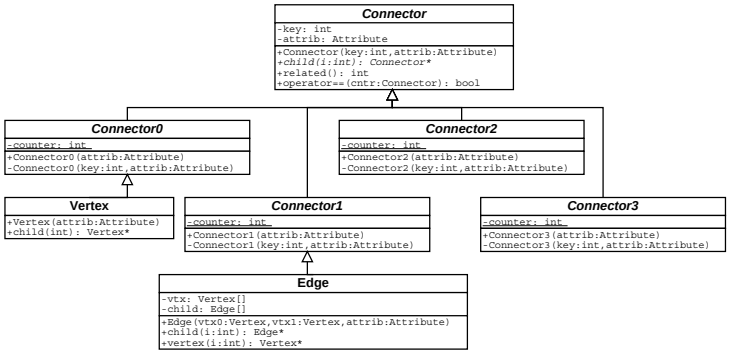
The hierarchy of the topological classes is shown in Figure 6.4. An important part of the topological information is knowledge about a cell's neighbours. However, this information is only stored implicitly in the data structure. A Quad stores links to its four edges and an Edge stores links to its two vertices. If two quadrilaterals share an edge, the edge only exists once in memory. To find out which two quadrilaterals share an edge, one has to loop over all quadrilaterals and check their edges. Algorithm 6.2 does this for a two dimensional mesh: it builds a (temporary) data structure containing references to all edges and the links to the cells belonging to the respective edges. In the implementation of this algorithm and others in this chapter, data structures of the Standard Template Library STL [86, 102] allowing efficient access to the data, are used.

Figure 6.5 shows the Object Diagram of a three dimensional mesh.

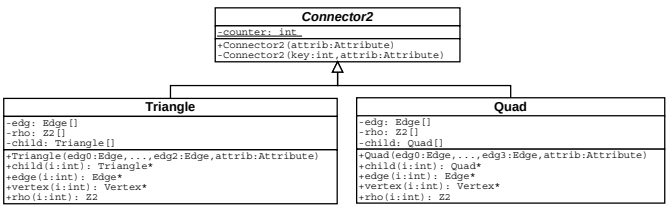
6.2.2 Subdivision Strategies

In Chapter 2, it was shown that anisotropic refinements are necessary for exponential convergence of *hp*-FEM. Restricting ourselves to refining an edge into two equally sized parts[3], the refinements in Figure 3.14 on page 79 need to be possible for a quadrilateral.

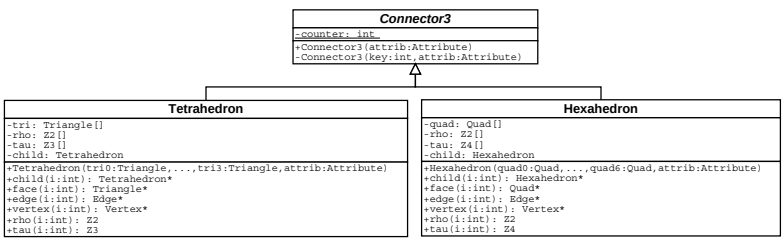
[3]The subdivisions ratio or geometric grading factor is $\sigma = 1/2$.



(a) Basic topological classes including zero and one dimensional specialisations Vertex and Edge respectively.



(b) Two dimensional specialisation of the topological classes: Triangle and Quad.



(c) Three dimensional specialisations of the topological classes: Tetrahedron and Hexahedron. The additional specialisations Prism and Pyramid are not shown here.

Figure 6.4: Classes for meshes II. A mesh consists of several cells: in one dimension of Edge1d and in three dimension of Tetrahedron3d and Hexahedron3d. Each of these cells has a reference to a topological entity and a reference to an element map. The class diagrams above show the hierarchy of the topological classes. The member counter in *Connector0–Connector3* is a static variable responsible for the generation of unique identifiers (the key) per spatial dimension. Z2, Z3 and Z4 are additive groups with two, three and four elements respectively used to describe orientation flags.

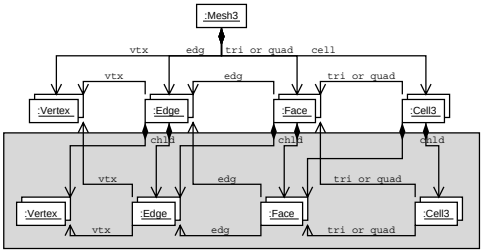


Figure 6.5: Objects in a mesh. The objects of a three dimensional mesh have the shown relations. If at least one of the cells is refined, the grayed part is added.

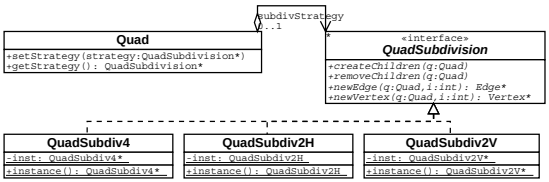


Figure 6.6: Subdivision strategies for a quadrilateral. The class Quad was already introduced in Figure 6.4 but not all members were shown there. Here, only the 'new' members are depicted.

This is achieved by applying the Strategy Pattern [56] to the subdivision algorithm in Quad (*c. f.* Figure 6.6). To remain backwards compatible (there was only the possibility to subdivide into four new quadrilaterals before), QuadSubdiv4 is the default.

Calling setStrategy with the appropriate argument changes the strategy in Quad. This has to be done before asking for the children of Quad (such a call for a child subdivides a cell if it has not yet been subdivided). However, if there already exist children of Quad, trying to change the subdivision strategy throws an exception. This exception is caught and handled correctly, the program does not abort but goes on.[4] However, the subdivision strategy and the children remain unchanged.

[4]Here, exceptions are not used to let the program fail but to indicate a special (*i. e.* exceptional) situation. However, precautions are taken to handle this situation (and cath the exceptions).

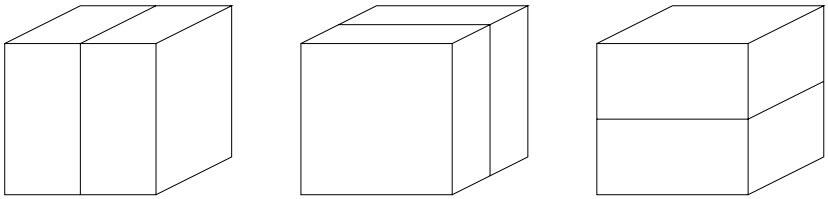


Figure 6.7: Refinement in directions ξ_1, ξ_2 and ξ_3.

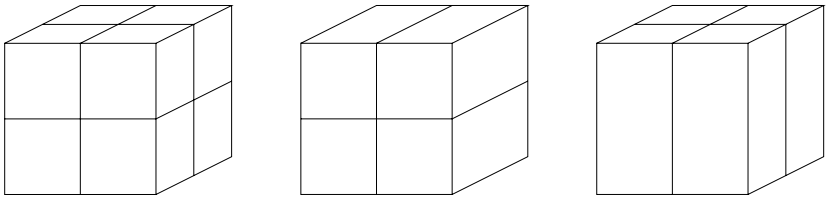


Figure 6.8: Examples of refinements resulting from combinations of refinements in Figure6.7.

The different refinements for a hexahedron are handled similarly (compared to three different strategies for the quadrilateral, there are seven for the hexahedron, examples are depicted in Figures 6.7 and 6.8). There is the additional technicality that the faces of the hexahedron need to be refined in the correct way in order to create the children correctly. This is solved by asking for the right subdivision strategy on the faces (which are quadrilaterals). If one of the faces cannot be refined in the necessary way (because of a different refinement strategy which was used previously for this face), an exception is thrown—with the same effects as in the case of the quadrilateral above.

Remark 6.3 (Geometric deadlock problem) *The handling of some combinations of refinements in neighbouring elements is not implemented (c. f. Figure 6.9). Nevertheless, they can be asked for—the result is an exception. The implementation of means to handle the combination shown in Figure 6.9 would not be straight forward. Furthermore, it is possible to refine one of the two elements to get a combination which is easier to handle (c. f. Figure 6.10).*

In the case shown in Figure 6.9, the main problem arises on the face which is shared by the two hexahedra. There, the identification of a 'first' and a 'second'

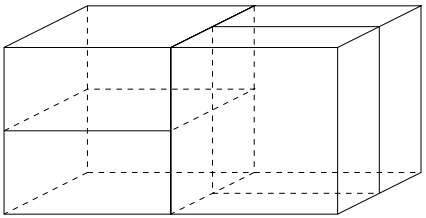


Figure 6.9: Illegal subdivision combination in neighbouring elements.

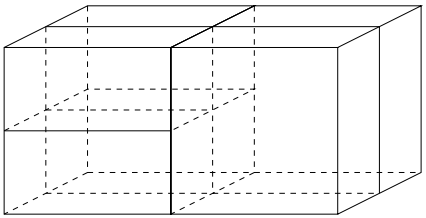


Figure 6.10: Legal subdivision combination in neighbouring elements.

child is ambiguous. The handling depends on these denotations, though. This problem is explained in more details in Section 7.2.

6.3 *hp*-Discretisation of $H^1_{\Gamma_D}(D)$

For *hp*-FEM, $\mathcal{S}^{\boldsymbol{p},1}_{\Gamma_D}(D,\mathcal{T})$ with $D \subset \mathbb{R}^d$ bounded ($d = 2$ or 3) has to be implemented. Here,

$$\mathcal{S}^{\boldsymbol{p},1}_{\Gamma_D}(D,\mathcal{T}) := \left\{u \in H^1_{\Gamma_D}(D)^d : u|_K \circ F_K \in \mathcal{V}_{\boldsymbol{p}_K} \quad \forall K \in \mathcal{T}\right\} \text{ and} \qquad (6.4)$$
$$H^1_{\Gamma_D}(D)^d := \left\{u \in H^1(D)^d : u|_{\Gamma_D} = 0\right\}.$$

To be able to represent all meshes described in Sections 2.2 and 2.3, anisotropic refinements (*h*-refinements) and an anisotropic polynomial degree vector $\boldsymbol{p} = \{\boldsymbol{p}_K\}_{K\in\mathcal{T}}$ need to be possible. The notation $\mathcal{T}$ and $\mathcal{V}_{\boldsymbol{p}_K}$ in (6.4) is explained below.

In this section, the implementation of $\mathcal{S}^{\boldsymbol{p},1}_{\Gamma_D}(D,\mathcal{T})$ in Concepts is described. The mathematical techniques and requirements were given in Chapter 3. The aim of the implementation of the FE space is a list of elements. Every element contains all necessary information (like geometry, assembling [T matrix] and shape functions). The object representing the space provides methods to loop over all elements in this list and to refine selected elements anisotropically in *h* and *p*.

This section is subdivided as follows: Firstly, the notion of the mesh $\mathcal{T}$ and the polynomial degree are detailed. The next part explains the shape functions used in one, two and three dimensions. Then, the support of a global basis function and how it is found are described. The central part describing the building of the

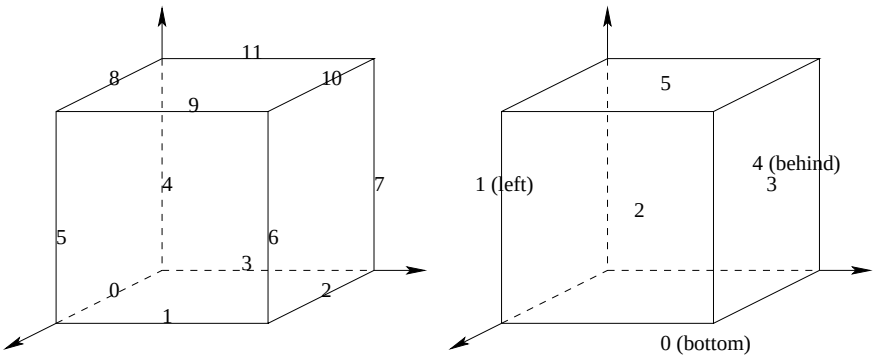


Figure 6.11: Numbering of the edges (left) and faces (right) in a hexahedron.

elements and the T matrices etc. follows last. A run-time cost analysis of the most expensive parts is given in Sections 3.3.6 and 3.4.

6.3.1 Specification of Mesh $\mathcal{T}$ and Polynomial degree p

Mesh $\mathcal{T}$

The classes to handle a mesh are described in Section 6.2.

Let $\mathcal{T}^1$ be a conforming, hexahedral[5] mesh of D (sometimes referred to as the coarsest mesh, the superscript ^{1} refers to the number of layers in the mesh). Then, $\mathcal{T}$ has to be a hierarchical refinement of $\mathcal{T}^1$. A refinement of a hexahedral element can be in direction ξ_1, ξ_2 or ξ_3 or a combination of these. Using such mesh refinements, $\mathcal{T}$ no longer needs to be conforming. The hanging nodes are handled with S matrices during the assembling procedure (*c. f.* Section 3.3.3).

Polynomial degree p

The notation $\mathcal{V}\boldsymbol{p}_K$ in (6.4) is described. $\boldsymbol{p}_K$ is the polynomial degree on element $K \in \mathcal{T}$. It has the following structure:

$$\boldsymbol{p}_K := \left\{ \boldsymbol{p}_K^c \in \mathbb{N}^3,\ \boldsymbol{p}_K^{f_i} \in \mathbb{N}^2 \text{ for } i = 0, \ldots, 5,\ p_K^{e_j} \in \mathbb{N} \text{ for } j = 0, \ldots, 11 \right\}, \tag{6.5}$$

[5]Concepts is also able to handle other element types (such as tetrahedra, prisms and pyramids). They are not covered in this work.

where c, f and e stand for *cell*, *face* and *edge* respectively. The numbering of the faces and edges of an element is given in Figure 6.11. Further on, K in $\boldsymbol{p}_K^c$, $\boldsymbol{p}_K^{f_i}$ or $p_K^{e_j}$ is omitted if it is clear from the context.

The following assertions hold for $\boldsymbol{p}_K$:

- $p \geq 1$ for all components p of $\boldsymbol{p}_K$.[6]
- If K and K' share a face or an edge, the polynomial degree on this face or edge is the same in both elements.[7]
- The polynomial degree on a child of a cell, face or edge is smaller or equal than the polynomial degree on the cell, face or edge itself, *i. e.* parents have larger or equal polynomial degree than children.

Now, $\mathcal{V}\boldsymbol{p}_K$ from (6.4) can be defined:

$$\begin{aligned}
\mathcal{V}\boldsymbol{p}_K := & \underbrace{\mathcal{P}^\star_{p_0^c} \otimes \mathcal{P}^\star_{p_1^c} \otimes \mathcal{P}^\star_{p_2^c}}_{\text{interior}} + \\
& \underbrace{\mathcal{P}^\star_{p_0^{f_0}} \otimes \mathcal{P}^\star_{p_1^{f_0}} \otimes \frac{1-\xi_3}{2}}_{\text{face } f_0} + \underbrace{\mathcal{P}^\star_{p_0^{f_1}} \otimes \frac{1-\xi_2}{2} \otimes \mathcal{P}^\star_{p_1^{f_1}}}_{\text{faces } f_i,\, i=1,\ldots,5} + \cdots + \\
& \underbrace{\mathcal{P}^\star_{p^{e_0}} \otimes \frac{1-\xi_2}{2} \otimes \frac{1-\xi_3}{2}}_{\text{edge } e_0} + \underbrace{\frac{1+\xi_1}{2} \otimes \mathcal{P}^\star_{p^{e_1}} \otimes \frac{1-\xi_3}{2}}_{\text{edges } e_i,\, i=1,\ldots,11} + \cdots + \\
& \underbrace{\frac{1-\xi_1}{2} \otimes \frac{1-\xi_2}{2} \otimes \frac{1-\xi_3}{2}}_{\text{vertices}} + \ldots
\end{aligned}$$

where $\mathcal{P}^\star_p := \frac{1-\xi^2}{4} \cdot \mathcal{P}_p$ and $\mathcal{P}_p = \text{span}\{\xi^i : 0 \leq i \leq p\}$. It is assumed that the reference element is $\hat{K} = (-1,1)^3$. The terms $\frac{1-\xi}{2}$ and $\frac{1+\xi}{2}$ originate from the linear shape functions, *c. f.* (6.8).

[6] It is possible to have piece-wise constant elements ($p = 0$) in Concepts, for instance for Boundary Element Methods, but this is not considered in this work.

[7] In fact, the polynomial degree on this face or edge is only stored once for both elements. For the faces, this involves handling the different possible orientations of the face which lead to a permutation of $p_0^{f_i}$ and $p_1^{f_i}$. This has to be taken care of in the implementation but will not be considered further here.

Remark 6.6 *$\boldsymbol{p}^c \in \mathbb{N}^3$ can be chosen arbitrary. $\boldsymbol{p}^{f_i} \in \mathbb{N}^2$ and $p^{e_j} \in \mathbb{N}$ can be raised if necessary (enriching the Finite Element space on the faces or edges) but not chosen arbitrary—the reason is explained in Section 6.3.3.*

Remark 6.7 (Trunk spaces) *It has been shown (e. g. [48, 49, 106]) that it helps save degrees of freedom to restrict the total polynomial degree in the interior and on the faces of a hexahedron. These so-called* trunk spaces *are also implemented in Concepts. There is nearly no sacrifice in accuracy in typical applications with respect to the full tensor product space, though. However, the amount of saved degrees of freedom is considerable: for $p = 8$ the full tensor product space in one hexahedron has 729 degrees of freedom and the trunk space has only 346. Figure 6.12 shows a comparison of the sparsity patterns. The plots were done with the following constraint polynomial spaces:*

$$\operatorname{span}\left\{x^j y^k : j + k \leq \max\left\{p_0^{f_i}, p_1^{f_i}\right\} + 1\right\}$$

on the faces and

$$\operatorname{span}\left\{x^i y^j z^k : i + j + k \leq \max\{p_0^c, p_1^c, p_2^c\} + 1\right\}$$

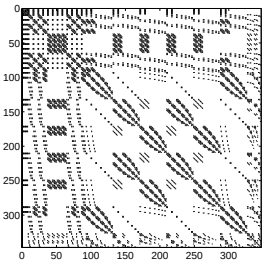
in the interior. In general, one could think of the following constraint polynomial spaces: $\{x^j y^k : f^f(j, k, \boldsymbol{p}^{f_i}) = 1\}$ and $\{x^i y^j z^k : f^c(i, j, k, \boldsymbol{p}^c) = 1\}$ with indicator functions f^f and f^c for the faces and the interior respectively. In fact, it would not be hard to add such an interface for user supplied functions f^f and f^c.

6.3.2 Shape Functions

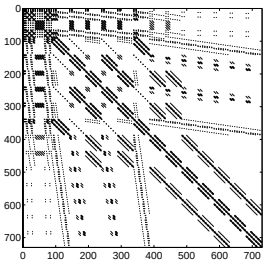
The reference element in which the shape functions are defined is $(-1, 1)^3$. The shape functions currently implemented are those proposed in [72]. They can easily be exchanged as long as the basis functions remain hierarchic and contain the first two linear shape functions.

In one dimension, the shape functions of order p (*c. f.* Figure 6.13) are

$$N_i(\xi) = \begin{cases} \frac{1-\xi}{2} & i = 0, \\ \frac{1+\xi}{2} & i = 1, \\ \frac{1-\xi}{2}\frac{1+\xi}{2} P_{i-2}^{1,1}(\xi) & 2 \leq i \leq p. \end{cases} \tag{6.8}$$



(a) Trunk space. Size: 346 × 346, number of non-zeros: 9528 (8.0%).



(b) Tensor product space. Size: 729 × 729, number of non-zeros: 18533 (3.5%).

Figure 6.12: Sparsity pattern of stiffness matrices in three dimensions for $p = 8$. The full tensor product stiffness matrix is sparser but twice as large (in size and number of non-zeros) compared to the matrix of the trunk space. The sparsity patterns themselves depend on the choice of the shape functions (see below). These plots here should illustrate the effect of the trunk spaces.

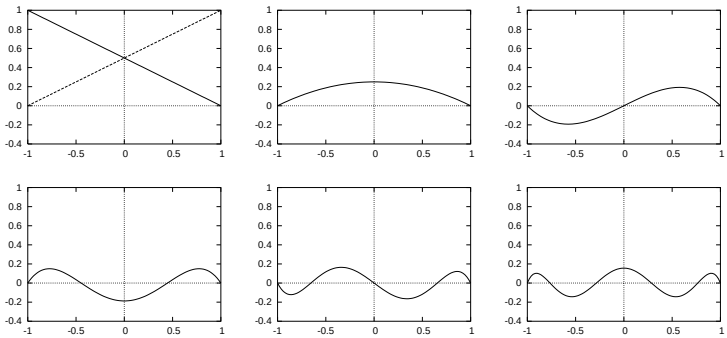


Figure 6.13: Shape functions (6.8) in one dimension for $p = 1, \ldots, 6$.

The Jacobi polynomials $P_i^{1,1}(\xi)$ are integrated Legendre Polynomials: $L_i(\xi) = P_i^{0,0}(\xi)$ and

$$\int_{-1}^{\xi} (1-x)^{\alpha}(1+x)^{\beta} P_i^{\alpha,\beta}(x)\,dx = \frac{-1}{2i}(1-\xi)^{\alpha+1}(1+\xi)^{\beta+1} P_{i-1}^{\alpha+1,\beta+1}(\xi)$$

$$\Rightarrow \int_{-1}^{\xi} P_i^{0,0}(x)\,dx = \frac{-1}{2i}(1-\xi)(1+\xi) P_{i-1}^{1,1}(\xi),$$

i. e. the shape functions given in (6.8) are essentially the integrated Legendre polynomials.

In higher dimensions, the shape functions are tensorised one dimensional shape functions. In two dimensions, this is done for the quadrilateral elements and in three dimension for the hexahedral elements.

The reason for choosing the shape functions (6.8) is the good sparsity pattern of the mass and stiffness matrices and the good behaviour of the condition number. Figure 6.14 shows the sparsity patterns of the mass and stiffness matrices of one element corresponding to the bilinear forms

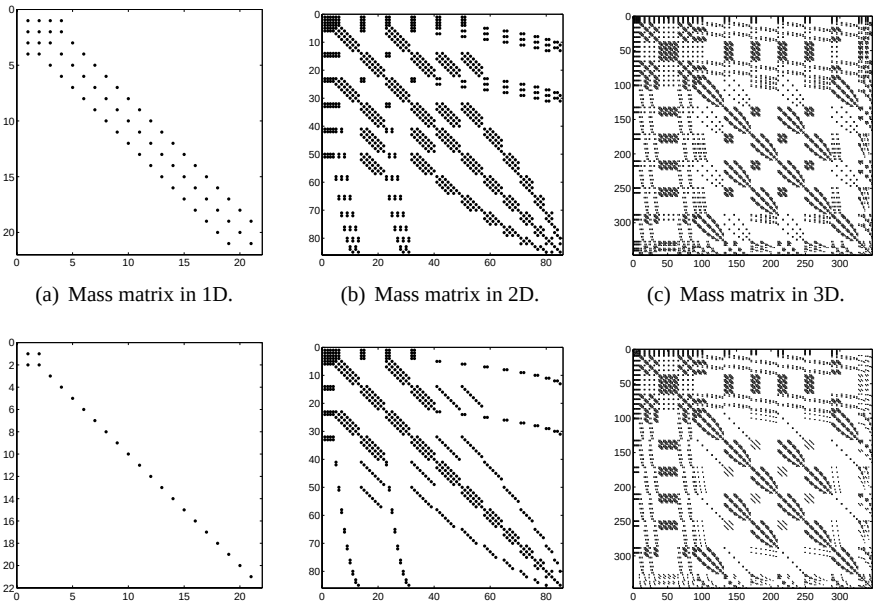
$$i(u,v) = \int_K uv\,d\boldsymbol{x} \qquad \text{(mass matrix)},$$

$$a(u,v) = \int_K \nabla u \cdot \nabla v\,d\boldsymbol{x} \qquad \text{(stiffness matrix)}$$

in one, two and three dimensions with internal and external degrees of freedom. Figure 6.15 shows the condition numbers of the mass and stiffness matrices corresponding to the above bilinear forms of one element in one, two and three dimensions with homogeneous Dirichlet boundary conditions on the whole of ∂D (*i. e.* only the internal degrees of freedom are considered). Again, the trunk spaces give better results (*i. e.* lower condition number).

6.3.3 Support of a Basis Function

Since every Finite Element function $u \in V_N = \mathcal{S}^{\boldsymbol{p},1}_{\Gamma_D}(D,\mathcal{T})$ is also in $H^1(D)$, it has to be made sure that u is continuous over element boundaries. This is automatically fulfilled by enforcing continuity for all basis functions of V_N. A key component of this task is to find the correct support for each basis function. The basis function is then combined from the individual shape functions in the cells of the support.



(a) Mass matrix in 1D. (b) Mass matrix in 2D. (c) Mass matrix in 3D.

(d) Stiffness matrix in 1D. (e) Stiffness matrix in 2D. (f) Stiffness matrix in 3D.

Figure 6.14: Sparsity pattern of the mass and stiffness matrices. The approximation orders are $p = 20$ in one dimension, $p = 10$ in two dimensions and $p = 8$ in three dimensions. In two and three dimensions, the total polynomial degree on the faces and in the interior was limited (*i. e.* trunk spaces).

In a complicated, locally refined mesh, it is not a-priori clear what the support of a certain basis function is (for instance corresponding to a vertex or an edge). This section shows how the support can be found and how the polynomial degree has to be adjusted in that support.

Finding the Support

To ensure global continuity of the basis function Φ_i, the unisolvent sets on the interfaces in the support of Φ_i have to match.

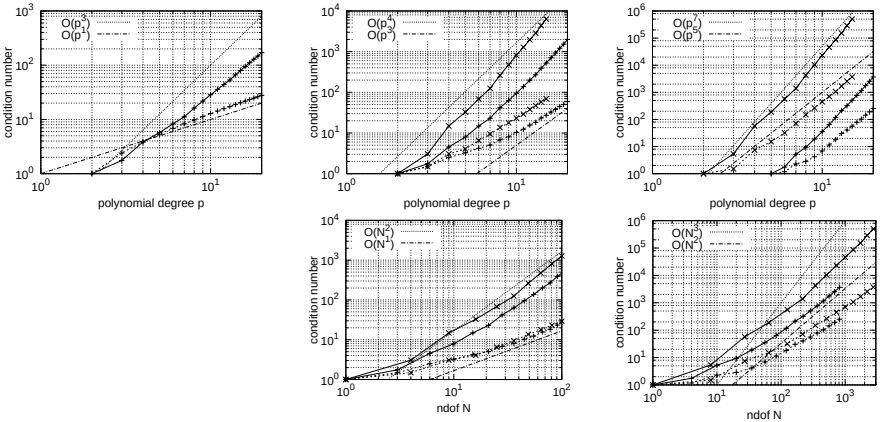


Figure 6.15: Condition numbers of the stiffness (dashed line) and mass matrices (solid line) in one, two and three dimensions (first, second and third column) plotted versus the polynomial degree p (top row) or the number of degrees of freedom N (short 'ndof', bottom row) using log-log scale. The lines marked with $+$ are computed using trunk spaces whereas $\times$ uses the full tensor product spaces.

	quadrilateral(two dimensional)	hexahedron(three dimensional)
on every edge	$p-1$	$p-1$
on every face		$(p-1)^2$
in the interior	$(p-1)^2$	$(p-1)^3$

Table 6.1: Points in the unisolvent sets for $\mathcal{Q}_p$ in addition to the vertices of the quadrilateral and hexahedron respectively.

- The unisolvent sets for $\mathcal{Q}_1$ (polynomials of maximal order one[8]) in a quadrilateral (two dimensional) or a hexahedron (three dimensional) are the corners of the quadrilateral and the hexahedron respectively.

- The unisolvent sets for $\mathcal{Q}_p$ are additional points on edges, faces and the interior (summarised in Table 6.1).

[8] In two dimensions: $\mathcal{Q}_p := \text{span}\{x^i y^j : i, j \leq p\}$, *i. e.* $\mathcal{Q}_1 = \text{span}\{1, x, y, xy\}$.

- Start with a list of all the smallest cells (from the 'active' level in the mesh $\mathcal{T}$) which contain the vertex v.
- Repeat until no changes have to be made:
  - Generate a list of all the edges which coincide in the vertex v and belong to one of the cells in the list.
  - Look for related edges in the prepared list. If two edges e and $\acute{e}$ are related (*i. e.* have an ancestor–descendant relationship) but are not equal:
    Replace the cell belonging to the smaller edge (without loss of generality denoted with e') with its ancestor $e^\star$ in such a way that the corresponding edges of e and $e^\star$ are equal, *i. e.* take the father, grandfather or great grandfather of $\acute{e}$ as $e^\star$.

Algorithm 6.3: Finding the support of a vertex mode corresponding to vertex v.

To find the support of a vertex mode, it suffices to match the edges which coincide in this vertex. To find the support of an edge mode (in three dimensions), the coinciding faces have to be matched.

Therefore, the idea of an algorithm to find a support of a vertex mode should be as shown in Algorithm 6.3. Finding the support of an edge mode in three dimensions follows the same lines as Algorithm 6.3—with faces replacing edges.

Example 6.9 (Finding the support of a vertex mode) *Figure 6.16 shows how Algorithm 6.3 finds the support of the bilinear basis function corresponding to middle vertex in four steps.*

The jump from the third to the fourth figure in Figure 6.16 is closely related to the handling of hanging nodes and the S matrices in Chapter 3. It is assumed that the top left quadrilateral in Figure 6.16 was refined at once into four smaller quadrilaterals. Therefore, the parent of a small quadrilateral is the parent to all four quadrilaterals in the algorithm to find the support of the basis function and also in the algorithm to handle hanging nodes.

Adjusting the Polynomial Degree

Typically, the degress of the faces and edges ($\boldsymbol{p}^f$ and p^e respectively) are determined by "projecting" the degree of the cell $\boldsymbol{p}^c$ onto the faces and edges respectively. Even in very simple cases in two dimensions, this can lead to problems.

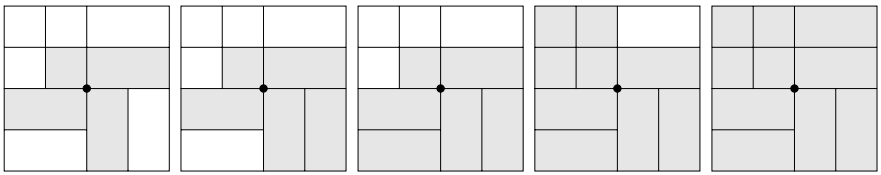


Figure 6.16: Example for Algorithm 6.3. It is shown how the support of the bilinear basis function corresponding to the vertex marked by • is found in four steps. The current list of the cells in the support of the basis function is grayed. The edges on the right of • are checked first. In this example, the algorithm goes on clockwise. The algorithm does not need or create a specific order, though.

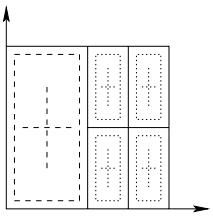


Figure 6.17: A dashed line means $p = 2$, a dotted line means $p = 1$: the left element has $p = 2$ in the interior (indicated by the cross) and on the edges and left four elements have $p = 1$ in the interior and on the edges.

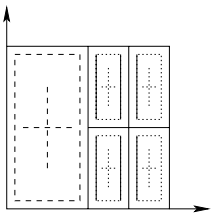


Figure 6.18: The edges marked with solid lines need to have $p = 2$ in order to be able to represent the shape function on the large middle edge.

Example 6.10 *Figure 6.17 shows a setup in* $(0,1)^2$ *with five elements and a polynomial degree of 2 in the large element and 1 in the small elements.*

The edge $\{1/2\} \times (0,1)$ *has one basis function* Φ *associated to it (namely the one of the edge of the large element with* $p = 2$*). The basis functions of the edges of the small elements are not taken into account since they are constrained by* Φ*. The basis function*

$$\Phi(x,y) = \begin{cases} 2x \cdot y(1-y) & x \le 1/2 \\ 2(1-x) \cdot y(1-y) & x \ge 1/2 \end{cases}$$

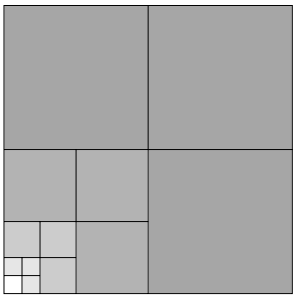


Figure 6.19: Geometrically refined mesh. Darker elements have a higher polynomial degree.

can be represented in the left element. Although the support of Φ *is* $(0,1)^2$*, it cannot be represented in any of the small elements: The polynomial degree of the edges on* $\{1/2+\} \times (0,1)$*,* $\{3/4-\} \times (0,1)$ *and* $\{3/4+\} \times (0,1)$ *is too small (only one instead of two). The problem can be resolved by dropping* Φ *from the Finite Element space by setting the polynomial degree of the edge* $\{1/2\} \times (0,1)$ *to 1 in all adjacent elements.*

The problem in Example 6.10 does not seem grave at first sight: one basis function is lost since it is not representable on all elements of the FE space. But in geometrically refined meshes (*c. f.* Figure 6.19) with linearly increasing polynomial degree, many basis functions would be lost, if this approach was followed. However, to ensure exponential convergence, these basis functions are essential. Since the supports of the basis functions on the edges with hanging nodes do also include the smallest elements with $p = 1$ (which do not include any shape functions on edges) all basis functions on edges with adjacent cells whose neighbours are not equally sized would have to be dropped.

A remedy for this problem would be to

- *shrink the support* of the concerned basis functions and raise the polynomial degree on the adjacent edges or
- *increase the polynomial degree* of all edges which are in the support of the concerned basis functions up to a sufficient level.

The second option is simpler to implement for the same reason as explained in Example 6.9. Namely, a subdivision into four children cannot easily be torn apart and regrouped: it cannot be reformulated as a subdivision into two children (followed by a second subdivision into two children). Therefore, shrinking the support of a basis functions would be very complicated.

Example 6.11 (Example 6.10 continued) *The degrees of the edge modes which were named in Example 6.10 have to be raised to 2 (c. f. Figure 6.18). Then, the basis function on the edge* $\{1/2\} \times (0, 1)$ *can be represented in its whole support* $(0, 1)^2$ *and is therefore a valid basis function of the FE space.*

6.3.4 Algorithm to Assemble the Finite Element Space

In Concepts, it is not possible to build a Finite Element space $\mathcal{S}^{\boldsymbol{p}',1}_{\Gamma_D}(D, \mathcal{T}')$ directly on an arbitrary mesh $\mathcal{T}'$ with an arbitrary polynomial degree distribution $\boldsymbol{p}'$. Instead, the adjustments for the mesh and the polynomial degree with respect to a FE space $\mathcal{S}^{\boldsymbol{p},1}_{\Gamma_D}(D, \mathcal{T})$ have to be given element-wise. An adjustment for an element $(K, \boldsymbol{p}_K)$ is a vector $\delta\boldsymbol{l} = (\delta l_x, \delta l_y, \delta l_z)$ and a vector $\delta\boldsymbol{p} = (\delta p_x, \delta p_y, \delta p_z)$ describing the refinements in each direction ($\delta\boldsymbol{l}$) and the increase in the polynomial degree ($\delta\boldsymbol{p}$). Here, x, y and z denote local coordinates relative to the element. This makes it possible to create a sequence of FE spaces. The starting point of such a sequence is the FE space $\mathcal{S}^{1,1}_{\Gamma_D}(D, \mathcal{T}^1)$ with polynomial degree 1 on the initial (conforming) mesh $\mathcal{T}^1$.

Building $\mathcal{S}^{\boldsymbol{p}',1}_{\Gamma_D}(D, \mathcal{T}')$ from $\mathcal{S}^{\boldsymbol{p},1}_{\Gamma_D}(D, \mathcal{T})$ takes these steps:

1. Determine $\mathcal{T}'$ and $\boldsymbol{p}'$ from $\mathcal{T}$ and $\boldsymbol{p}$ and the adjustments $\delta\boldsymbol{l}$ and $\delta\boldsymbol{p}$. Enforce Dirichlet boundary conditions by marking the respective modes as 'passive' (explained below).

2. Find the support of all basis functions on vertices, edges and faces (*c. f.* the first part of Section 6.3.3).

3. Enforce the minimum rule: the polynomial degree on a face or edge has to be the minimum of the adjacent elements. This economises a few degrees of freedom without harming the convergence.

4. Enrich the polynomial degree on some faces and edges (*c. f.* the second part of Section 6.3.3).

5. Build the elements of the FE space (including the computation of the T matrices and the application of the S matrices).

Each of these steps recursively traverses the mesh (which has a tree structure) in a depth first search fashion. The entry point for such a traversal is the first cell in $\mathcal{T}^1$. These steps are presented in the following subsections.

Determining Mesh and Polynomial Degree

Algorithm 6.4 shows how the adjustments describing the transition from $\mathcal{T}$ to $\mathcal{T}'$ and from $\boldsymbol{p}$ to $\boldsymbol{p}'$ are taken into account. meshAndPoly is called inside a loop over all cells c in the initial mesh $\mathcal{T}^1$. The initial $\boldsymbol{p}^\star$ (parameter of meshAndPoly) is set to $(-1, -1, -1)$. The purpose of meshAndPoly is to determine which cells are 'member of the space' and the polynomial degree of the cells. The polynomial degree on the edges and faces is fixed later.

The attribute 'member of the space' of a degree of freedom on a face or an edge which is set in (6.12) means that the degree of freedom belongs to the currently constructed FE space. This attribute is shared between elements which share this face or edge. Additionally, each degree of freedom can have an attribute 'passive' (opposed to 'active') which essentially excludes the degree of freedom from the FE space. This is used for homogeneous Dirichlet boundary conditions.

Finding the Support of Vertex, Edge and Face Modes

The key part of this task has already been described in the first part of Section 6.3.3. What remains to be explained is how it all fits together.

At the beginning, lists of vertices, edges and faces are created. Each of these lists is created by looping over the cells in the current mesh. The lists are made in a temporary data structure so that more data is at hand which is not stored explicitly in the mesh data structure, namely:

- parent pointers for the cells,
- pointers to the cells and edges for the vertices,
- pointers to the cells and faces for the edges and
- pointers to the cells for the faces.

- If $c \in \mathcal{T}$:
  - Adjust $\boldsymbol{p}^c$ on c and refine (or coarsen) c if necessary (depending on the *adjustments*, *i. e.* generate the children of c).
  - Ensure $\boldsymbol{p}^c \geq 1$ and set $\boldsymbol{p}^\star = \boldsymbol{p}^c$.
- If $c \notin \mathcal{T}'$:
  - *Call* meshAndPoly(c', $\boldsymbol{p}' := \boldsymbol{p}^\star$) *for all children* c' *of* c. After each call: $\boldsymbol{p}^\star = \max\{\boldsymbol{p}^\star, \boldsymbol{p}'\}$.
  - $\boldsymbol{p}^c = \boldsymbol{p}^\star$.
  - $\forall$ children c' of c (f_i' and e_j' are the faces and edges of c' respectively):

$$\boldsymbol{p}^{f_i} = \max\{\boldsymbol{p}^{f_i'}, \boldsymbol{p}^{f_i}\} \text{ for } i = 0, \ldots, 5, \text{ if } f_i' \text{ and } f_i \text{ are related}$$

$$p^{e_j} = \max\{p^{e_j'}, p^{e_j}\} \text{ for } j = 0, \ldots, 11, \text{ if } e_j' \text{ and } e_j \text{ are related.}$$

- else (*i. e.* $c \in \mathcal{T}'$):
  - Set $\boldsymbol{p}^c = \boldsymbol{p}^\star$ and project $\boldsymbol{p}^c$ onto the faces and edges of c.
  - Mark the degrees of freedom of the cell (including the faces and edges) as '*member of the space*'. (6.12)

Algorithm 6.4: meshAndPoly(c, $\boldsymbol{p}^\star$), where c is the cell and $\boldsymbol{p}^\star$ is the desired polynomial degree in c respectively. $\mathcal{T}$ is the mesh used for the old space and $\mathcal{T}'$ is the mesh used for the new space. Initially, $\boldsymbol{p}^\star = (-1, -1, -1)$.

This additional data is used in the algorithms to find the support for the modes on the vertices, edges and faces.

After the initial set-up phase, relations are checked in the list of faces following Algorithm 6.5. Algorithm 6.5 implements the same idea as found in Algorithm 6.3 but for the somewhat simpler case of faces. This ensures the global continuity of face modes.

The same procedure preformed for the faces is now done with the edges, *i. e.* Algorithm 6.5 is applied to the list of edges. However, this is not sufficient to ensure globally continuous edge modes.

- While anything changes
  - Look for pairs of faces f and f' with an ancestor–descendant relationship. If such a pair is found (without loss of generality f' is a descendant of f):
    - Add the ancestor of the cell of f' which has the same size as the cell of f to list of cells of f.
    - Remove f' from the list of faces.

Algorithm 6.5: Checks for related faces in the list of faces. Eventually, every interior face has two cells registered which have equal size (*i. e.* have the whole face as one of their sides).

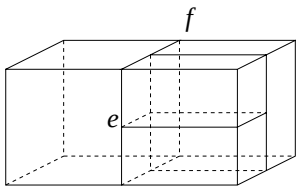


Figure 6.20: Illustrating the problem of the edge modes in Example6.13.

Example 6.13 (Not globally continuous edge mode) *Consider the mesh in Figure 6.20. After the initial set-up phase, the large cell on the left is the only cell being in the support of the basis functions corresponding the large, vertical edge e.*

After the applying Algorithm 6.5, edge e has the large cell on the left and the parent of the two cells in the foreground on the right in its support. This, however, does not define the support of a globally continuous edge mode on edge e as the unisolvent sets on the face f do not match.

The correct support for the edge e would be the whole depicted domain.

The problem described in Example 6.13 is fixed by applying an analogon to Algorithm 6.3 to every edge in the list of edges.

The last step in this phase is to find the correct support for the vertex modes. This is done by applying Algorithm 6.3 to every vertex in the list of vertices.

- If $c \in \mathcal{T}$:

$$\boldsymbol{p}^{f_i} = \max\left\{\boldsymbol{q}^{f_i}, \boldsymbol{p}^{f_i}\right\} \text{ for } i = 0, \ldots, 5,$$
$$\boldsymbol{p}^{e_j} = \max\left\{\boldsymbol{q}^{e_j}, \boldsymbol{p}^{e_j}\right\} \text{ for } j = 0, \ldots, 11,$$

  i. e. take the maximum of $\boldsymbol{p}$ and $\boldsymbol{q}$ on the boundary of the cell.

- else (*i. e.* $c \notin \mathcal{T}$): Loop over all children c' of c:
  - If there exist degrees of freedom on the boundary of c:
    - $\boldsymbol{q}^{f_i} = \max\left\{\boldsymbol{q}^{f_i}, \boldsymbol{p}^{f_i}\right\}$, $i = 0, \ldots, 5$, if the basis functions on f_i of c' are necessary to represent the basis functions on f_i of c (in c'). This heavily depends on the subdivision strategy and the position of f_i with respect to c and the faces of c.
    - Analogously for $\boldsymbol{q}^{e_j}$, $j = 0, \ldots, 11$.
  - *Call* enrichElm$(c', \boldsymbol{q})$.

Algorithm 6.6: enrichElm$(c, \boldsymbol{q})$, where c is the cell and $\boldsymbol{q}$ is the minimal polynomial degree c has to fulfil on its boundary ($\boldsymbol{q}$ has the structure (6.5)).

Enforcing the Minimum Rule

Algorithm 6.4 projects the internal polynomial degree $\boldsymbol{p}^c$ onto the faces and edges of c in an arbitrary order (given by the structure of the mesh $\mathcal{T}$). However, using the minimal polynomial degree of the adjacent cells as the polynomial degree on the edges and faces saves some degrees of freedom. This is done in a separate phase by looping over all faces and all edges in the list of faces and the list of edges respectively. On each face or edge, a loop over all adjacent cells is performed and the minimal polynomial degree is computed and registered with the face or edge.

Enriching the Polynomial Degree of Faces and Edges

Algorithm 6.6 shows how the polynomial degree of certain faces and edges is raised in order to represent the necessary basis functions. The reason for this was given in the second part of Section 6.3. Initially, enrichElm is called with $\boldsymbol{q} = (-1, \ldots, -1)$ on all cells of $\mathcal{T}^1$.

- $\forall$ vertices v of c:
  - If v is 'active' and 'member of the space' and c is in the support of the vertex mode corresponding to v: Get a global index for the degree of freedom, build a T column and add it to T. (6.14)
  - If v is 'passive' or 'member of the space' and c is not 'member of the space': *Deactivate* v, *i. e.* mark v as 'passive'.
- Analogously for the edges and faces. When computing the entries of the T column, the orientation and the polynomial degrees on the edges and faces are taken into account. Deactivating an edge or face means that all existing children of the edge or face are also deactivated (*i. e.* marked as 'passive').
- If $c \in \mathcal{T}$: Get the global indices for the inner degrees of freedom, build T columns for them and add the T columns to T. Eventually, build an element from c with T and add it to the list of elements of the space.
- If $c \notin \mathcal{T}$: $\forall c'$ children of c:
  - *Apply the S matrix* $\underline{S}_{cc'}$: $\forall \boldsymbol{t} \in T : \boldsymbol{t}' = \underline{S}_{cc'}\boldsymbol{t}$ and add $\boldsymbol{t}'$ to T'. (6.15)
  - *Call* buildElements(c', T').

Algorithm 6.7: buildElements(c, T), where c is the cell and T is a list of the T columns which are going to be integrated into the T matrix of c.

Building the Elements

Eventually, Algorithm 6.7 shows how the elements are built. The test if the cell c was in the support of the vertex mode at v in (6.14) uses the list of vertices which was built in the previous phases. Analogously, when building the T columns for the edges and faces, the lists of edges and faces are used.

(6.15) implements Proposition 3.29 column-wise. The T matrix $\underline{\tilde{T}}_{K'}$ of the basis functions in B_{insert} is empty as Algorithm 6.7 only computes the basis functions in B_{keep}.

6.3.5 Numerical Experiments

The algorithms presented in this section are not only suited for the meshes generated by Algorithm 3.3 but also for meshes guided by a-posteriori error estimation.

- Initialise the refinement indicators for the subdivision $\boldsymbol{l} = (0, 0, 0)$ and the increase of the polynomial degree $\boldsymbol{p} = (0, 0, 0)$.
- Separately for each entry of $\delta\boldsymbol{l}$ and $\delta\boldsymbol{p}$: set the entry to 1 if $r >$ RAND_MAX/2, where r is an integer computed by the pseudo-random number generator between 0 and RAND_MAX.
- Refine the element by $\delta\boldsymbol{l}$ and $\delta\boldsymbol{p}$.

Algorithm 6.8: Randomised refinement of one element.

A-posteriori error estimation might not generate the same evenly structured meshes as the a-priori defined refinements of Algorithm 3.3. This is simulated by the *randomised refinement*: Each element is refined with Algorithm 6.8 to generate a new hp-FE space.

We demonstrate that our hp-FE space generation algorithms work as expected also for random meshes by solving the reaction-diffusion problem

$$-\Delta u + u = f \text{ in } D, \qquad u = 0 \text{ on } \partial D,$$

with the two exact solutions

$$u(\boldsymbol{x}) = (x_1 - 1)x_1(x_2 - 1)x_2(x_3 - 1)x_3, \tag{6.16}$$

$$u(\boldsymbol{x}) = \sin(\pi x_1)\sin(\pi x_2)\sin(\pi x_3). \tag{6.17}$$

Both problems are analytic—(6.16) is even polynomial. The exact energies are computed by Mathematica.

The convergence history of the relative energy error is shown in Figure 6.21 together with time measurements for the space generation, the stiffness matrix computation and the solution of the linear system. Compared to the run-time cost analysis in Section 3.4, the space generation takes much more time in this case. However, the maximal polynomial degree used in the randomised examples shown here is 4 and a polynomial degree of 4 is still in the pre-asymptotic range of the run-time cost analysis in Section 3.4. There, one can also observe that the space generation takes more time than the stiffness matrix computation for low polynomial degrees (*c. f.* Figure 3.17). On the other hand, it is very likely that the generation of FE space with randomised refinement is more expensive than the generation of the FE

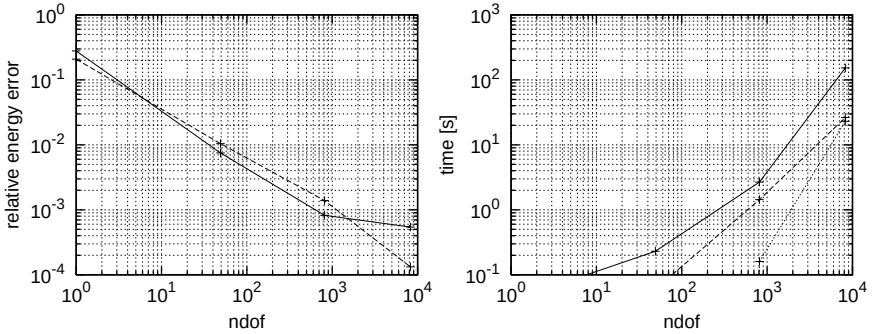


Figure 6.21: Randomised refinement with analytical exact solutions in $(0,1)^3$. The left plot shows the relative energy error plot versus the number of degrees of freedom: problem (6.16) (solid line) and problem (6.17) (dashed line). The right plot shows various time measurements versus the number of degrees of freedom: FE space generation (solid line), stiffness matrix computation (dashed line) and solution of the linear system (dotted line).

spaces used in Section 3.4 as more irregularities are present (more applications of S matrices) and also the search algorithms introduced above take more time.

6.4 Fast Integration of Element Matrices: Sum Factorisation

Besides solving the final linear system,[9] the computation of the element matrices is the most time consuming part of a FE solve. Therefore, it is beneficial to put some extra effort into speeding it up.

This section reviews the most general idea: sum factorisation. It only depends on the product structure of the shape functions. Further speed-ups can be gained from nodal shape functions [84] or hierarchical shape functions (*c. f.* Section 6.4.2 below), but these methods are both not considered in Concepts.

[9] This means that the time for the solution of the linear system is not taken into account here. It does not mean that it is the most time consuming part, though.

6.4.1 Theoretical Derivation

The integration in a hexahedral element K amounts to evaluating the sum

$$\left[\underline{A}\right]_{\boldsymbol{i}\,\boldsymbol{j}} = \sum_{q_{\{1,2,3\}}=1}^{n_{q_{\{1,2,3\}}}} N^1_{i_1}(\xi_{q_1}) M^1_{j_1}(\eta_{q_1}) N^2_{i_2}(\xi_{q_2}) M^2_{j_2}(\eta_{q_2}) N^3_{i_3}(\xi_{q_3}) M^3_{j_3}(\eta_{q_3}) \cdots$$
$$\cdots w_{q_1} w_{q_2} w_{q_3} b(\boldsymbol{\xi}, \boldsymbol{\eta}) \quad (6.18)$$

for each matrix entry $\boldsymbol{i}$, $\boldsymbol{j}$ where $\boldsymbol{i} = (i_1, i_2, i_3)$ and $\boldsymbol{j} = (j_1, j_2, j_3)$ are the matrix indices in tensor product notation. In (6.18), $N^k(.)$ and $M^l(.)$ are the reference element shape functions in direction k and l (one of 1, 2 or 3) respectively. ξ_{q_i} and η_{q_i} are the abscissas and w_{q_i} the weights of the quadrature rule for the numerical integration. $b(\boldsymbol{\xi}, \boldsymbol{\eta})$ summarises the coefficients of the PDE and the determinant of the Jacobian of the element map $F_K : \hat{K} \to K$.

For an isotropic polynomial degree p and $\mathcal{O}(p)$ quadrature points in each direction, the evaluation of (6.18) takes $\mathcal{O}(p^3)$ multiplications. In an element matrix, there are $\mathcal{O}(p^6)$ entries. All in all, an $\mathcal{O}(p^9)$ algorithm.

As shown in [84], a reordering of (6.18) to

$$\left[\underline{A}\right]_{\boldsymbol{i}\,\boldsymbol{j}} = \sum_{q_1=1}^{n_{q_1}} N^1_{i_1}(\xi_{q_1}) M^1_{j_1}(\eta_{q_1}) \sum_{q_2=1}^{n_{q_2}} N^2_{i_2}(\xi_{q_2}) M^2_{j_2}(\eta_{q_2}) \cdots$$
$$\cdots \sum_{q_3=1}^{n_{q_3}} N^3_{i_3}(\xi_{q_3}) M^3_{j_3}(\eta_{q_3}) w_{q_1} w_{q_2} w_{q_3} b(\boldsymbol{\xi}, \boldsymbol{\eta})$$

and the computation of the two temporary matrices

$$\left[\underline{R}\right]_{q_1,q_2,i_3,j_3} = \sum_{q_3=1}^{n_{q_3}} N^3_{i_3}(\xi_{q_3}) M^3_{j_3}(\eta_{q_3}) w_{q_1} w_{q_2} w_{q_3} b(\boldsymbol{\xi}, \boldsymbol{\eta}) \quad (6.19)$$

and

$$\left[\underline{S}\right]_{q_1,i_2,j_2,i_3,j_3} = \sum_{q_2=1}^{n_{q_2}} N^2_{i_2}(\xi_{q_2}) M^2_{j_2}(\eta_{q_2}) \cdot \left[\underline{R}\right]_{q_1,q_2,i_3,j_3} \quad (6.20)$$

results in a reduction to $\mathcal{O}(p^7)$ for the whole evaluation of

$$\left[\underline{A}\right]_{\boldsymbol{ij}} = \sum_{q_1=1}^{n_{q_1}} N_{i_1}^1(\xi_{q_1}) M_{j_1}^1(\eta_{q_1}) \cdot \left[\underline{S}\right]_{q_1,i_2,j_2,i_3,j_3}. \tag{6.21}$$

[84] shows more tricks to reduce the constant in $\mathcal{O}(p^7)$ and how to reduce the order further with different shape functions.

6.4.2 Hierarchical Shape Functions

This is from private communication with Joachim Schöberl at Johannes Kepler Universität in Linz, Austria [95].

The complexity of a standard element mass matrix computation is $\mathcal{O}(p^9) = \mathcal{O}(p^{3n})$ in three dimensions. This is reduced to $\mathcal{O}(p^7)$ by the sum factorisation algorithm shown above. A further reduction to $\mathcal{O}(p^6) = \mathcal{O}(p^{2n})$ is possible by exploiting the hierarchical structure of the shape functions.

Assume the hierarchical shape functions are defined by the recurrence relation

$$P_i(\xi) = a_i \xi P_{i-1}(\xi) + b_i P_{i-2}(\xi). \tag{6.22}$$

The three summations (6.19)–(6.21) above can be generalised and simplified to

$$\left[\underline{M}\right]_{ij} := \sum_q \rho(\xi_q) P_i(\xi_q) P_j(\xi_q).$$

Using the recursion formula (6.22) in i:

$$\begin{aligned}\left[\underline{M}\right]_{i+1,j} &= \sum \rho P_{i+1} P_j = \sum (a_{i+1}\xi_q P_i + b_{i+1} P_{i-1}) P_j \\ &= a_{i+1} \sum \rho \xi_q P_i P_j + b_{i+1} M_{i-1,j}\end{aligned}$$

and in j:

$$\left[\underline{M}\right]_{i,j+1} = a_{j+1} \sum \rho \xi_q P_i P_j + b_{j+1} M_{i,j-1}.$$

Combining both leads to the relation

$$\frac{1}{a_{i+1}} \left[\underline{M}\right]_{i+1,j} - \frac{b_{i+1}}{a_{i+1}} \left[\underline{M}\right]_{i-1,j} = \frac{1}{a_{j+1}} \left[\underline{M}\right]_{i,j+1} - \frac{b_{j+1}}{a_{j+1}} \left[\underline{M}\right]_{i,j-1}. \tag{6.23}$$

- Precompute the Jacobian and the other coefficients which are needed in all quadrature points.
- Set all entries in $\underline{A}$ to zero.
- Prepare the auxiliary objects u and v including the shape functions for all three directions and the respective quadrature rules.
- Initialise the accessor object of the array of coefficients.
- Add the contributions of u and v with the given coefficients into $\underline{A}$ by calling the *sum factorisation routine.*
- Return $\underline{A}$.

Algorithm 6.9: Mass matrix computation by using the sum factorisation routine.

By this identity, it is possible to fill in the values $[\underline{M}]_{ij}$. Only two rows must be evaluated explicitly.

Remark 6.24 *Here, the identity was derived based on the summation from the numerical integration. However, it is also possible to do so based on the integrals: The same relation* (6.23) *holds.*

This simple example can be extended to the case (6.19)–(6.21) *and could be applied there given hierarchical shape functions. It is also possible to derive a similar identity for other recurrence relations than* (6.22).

However, this is not considered in our implementation as the goal was to be independent of the concrete form of the shape functions.

6.4.3 Implementational Aspects

What was shown so far is valid for mass matrices (one evaluation of (6.18)) and stiffness matrices (nine evaluations of (6.18)). The same algorithm can also be used for the evaluation of the element matrices of other operators like the **curl-curl** operator in electromagnetics. All the differences of these evaluations go into $b(\boldsymbol{\xi}, \boldsymbol{\eta})$. This is also true in the implementation.

To make the routine computing the sum factorisation as reusable as possible, it is templated on the type of coefficients. The routine is called with a large number of precomputed arrays (up to 15 different arrays for the quadrature rule, the shape functions, the coefficients and the resulting element matrix). Two types of coeffi-

- Precompute the Jacobian and the other coefficients which are needed in all quadrature points.
- Set all entries in $\underline{\underline{A}}$ to zero.
- Prepare the auxiliary objects u and v including the shape functions for all three directions and the respective quadrature rules.
- For $i = 1$ to 3
  - Exchange the i^{th} component of u with the derivatives of the shape functions.
  - For $j = 1$ to 3
    - Exchange the j^{th} component of v with the derivatives of the shape functions.
    - Initialise the accessor object of the array of coefficient matrices with the indices i, j.
    - Add the contributions of u and v with the given coefficients into $\underline{\underline{A}}$ by calling the *sum factorisation routine.*
    - Restore v.
  - Restore u.
- Return $\underline{\underline{A}}$.

Algorithm 6.10: Stiffness matrix computation by using the sum factorisation routine.

cients are possible: scalars (in the case of a mass matrix) and 3×3 matrices (in cases like stiffness matrices, **curl-curl** matrices etc.). The scalar case is simple: every entry in the array of coefficients has to be used. In the case of a coefficient matrix, of all matrices in the array, only the entry i, j has to be used. This is managed by an accessor object. There should be a loop over all values of i, j outside the call to the sum factorisation routine (*c. f.* Algorithms 6.9 and 6.10). In Section 7.1.3, the bilinear form of the **curl-curl** operator is derived. The formulation there shows that an algorithm similar to Algorithm 6.10 has to be called nine times to compute an element matrix for the **curl-curl** operator. The sum factorisation routine, however, is the same.

6.4.4 Experiments

We try to experimentally confirm the run-time performance of the implementation of the sum factorisation routine by measurements of the stiffness matrix compu-

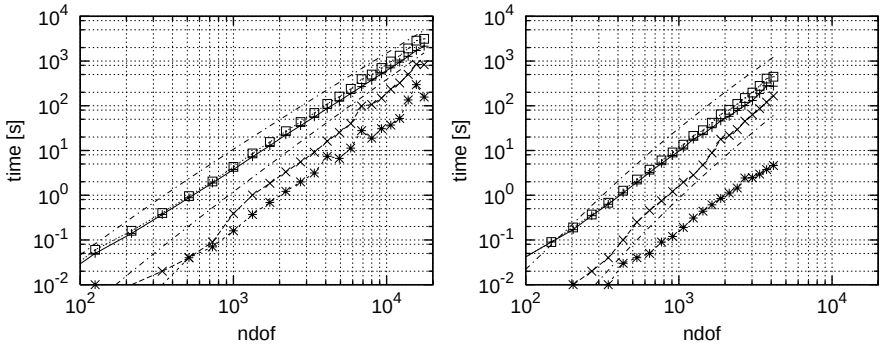


Figure 6.22: Timings for building a stiffness matrix for a mesh with one hexahedron with an isotropic polynomial degree up to 25. The left plot shows the full tensor product space and the right plot the trunk space (Remark 6.7). In both plots, the dash-dotted lines show p^6 and p^7. The other curves show the measured CPU times for the following tasks (from top to bottom): total time ($\square$), the computation of the element stiffness matrix ($+$), the assembly of the element matrix into the global matrix ($\times$) and the application of the T matrix ($*$).

tation time in one element with isotropic polynomial degree. Information on the CPU and compiler used and how the time measurements are conducted are given in Section 7.3.

Figure 6.22 does not experimentally confirm the work estimate in the previous section: the sum factorisation is an $\mathcal{O}(p^7)$ algorithm and what is shown is $\mathcal{O}(p^6)$. A more fine-grained timing of the parts in the sum factorisation routine is shown in Figure 6.23. However, there is nothing of order 7 visible. Apparently, the p^6 term dominates the p^7 term because of a larger constant.

Investigating the operation counts for the full tensor product space for polynomial degrees $p = 1, \ldots, 9$ reveals that the operation counts for (6.21) behave like

$$25p^7 + 373p^6 + 1724p^5 + 3911p^4 + 5111p^3 + 4048p^2 + 1861p + 394,$$

i. e. the $\mathcal{O}(p^7)$ is visible here. $373/25 \approx 15$ shows that the p^6 term dominates the p^7 term up to $p = 14$.

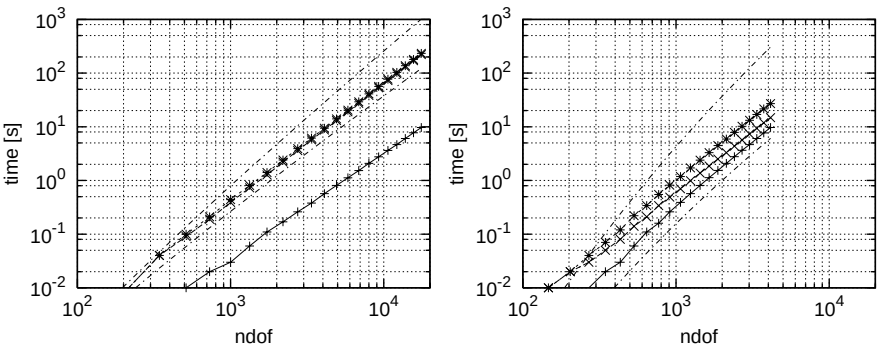


Figure 6.23: Timings on the element level for building a mass matrix in a hexahedron with an isotropic polynomial degree up to 25. The left plot shows the full tensor product space and the right plot the trunk space (Remark 6.7). In both plots, the dash-dotted lines show p^6 and p^7. The other curves show the measured CPU times for the following tasks (from top to bottom): total time for the sum factorisation routine ($*$), evaluation of (6.21) ($\times$) and evaluation of (6.20) ($+$).

7 Additional Matters

The last chapter of this part shows how the scalar spaces can be combined using a Cartesian product to discretise vector valued problems such as Maxwell's equations (*c. f.* Chapter 4), linear elasticity or Stokes' equations. In addition, a geometric deadlock problem and the implemented solution are presented.

7.1 Vector Valued Problems

This section explains the ideas and classes to build vector valued FE spaces from scalar FE spaces as needed for instance for Stokes' or Maxwell's equations and linear elasticity. The classical way to discretise Maxwell's equations using Nédélec's elements is not covered by this, though: only Cartesian products of scalar FE spaces are covered.

Remark 7.1 *Nonetheless, Nédélec type elements can be implemented in Concepts. However, one cannot rely on previously developed scalar Finite Element spaces. Instead, the vector valued elements have to be implemented directly. Then, a vector valued Finite Element space using them is constructed.*

This direct approach is also feasible for the vector valued Finite Elements which could be created using the Cartesian product Ansatz described below. However, there remains a considerable amount of work although this can be saved by using the ideas below.

7.1.1 Mathematics and Basic Ideas

Chapter 4 describes how Maxwell's equations can be discretised using nodal Finite Elements. Eventually, a FE space $\mathcal{S}^{\boldsymbol{p},1}_{\Gamma_D}(D,\mathcal{T})^3$ has to be discretised. Lin-

ear elasticity in two dimensions needs a FE space like $\mathcal{S}^{\boldsymbol{p},1}_{\Gamma_D}(D,\mathcal{T})^2$. In both cases, it might occur that the boundary conditions are different in the different components of the Cartesian product, *i. e.* one needs to have a FE space like $\mathcal{S}^{\boldsymbol{p},1}_{\Gamma_{D_1}}(D,\mathcal{T}) \times \mathcal{S}^{\boldsymbol{p},1}_{\Gamma_{D_2}}(D,\mathcal{T})$. All this is covered by the following *Cartesian product* of FE spaces.

The framework vectorial has the following properties:

1. The mesh $\mathcal{T}$ is the same for all scalar spaces in the product.
2. The user is able to reuse scalar spaces, bilinear and linear forms.
3. The classes for assembling, matrices, vectors, solvers, meshes and graphics are reused.
4. There is no limit on the number of factors in the Cartesian product (except the available memory).

Basis of the Vector Valued Finite Element Space

Mathematically, a FE space consist of a finite set of basis functions with a bounded support. The basis functions in a product space with two factors can be constructed in the following way:

$$\begin{aligned} V &= \operatorname{span}\left\{\Phi^V_1,\ldots,\Phi^V_n\right\},\\ W &= \operatorname{span}\left\{\Phi^W_1,\ldots,\Phi^W_m\right\}\\ \Rightarrow V\times W &= \operatorname{span}\left\{\begin{pmatrix}\Phi^V_1\\0\end{pmatrix},\ldots,\begin{pmatrix}\Phi^V_n\\0\end{pmatrix},\begin{pmatrix}0\\\Phi^W_1\end{pmatrix},\ldots,\begin{pmatrix}0\\\Phi^W_m\end{pmatrix}\right\}. \end{aligned} \tag{7.2}$$

The ordering of the basis functions in $V \times W$ in (7.2) is such that the resulting system matrices and vectors have block structure:

$$\underline{A} = \begin{pmatrix}\underline{A}^{VV} & \underline{A}^{VW}\\ \underline{A}^{WV} & \underline{A}^{WW}\end{pmatrix}, \boldsymbol{l} = \begin{pmatrix}\boldsymbol{l}^V\\ \boldsymbol{l}^W\end{pmatrix}. \tag{7.3}$$

The sizes of the matrices are as follows: $\underline{A}^{VV} \in \mathbb{R}^{n\times n}$, $\underline{A}^{VW} \in \mathbb{R}^{n\times m}$, $\underline{A}^{WV} \in \mathbb{R}^{m\times n}$ and $\underline{A}^{WW} \in \mathbb{R}^{m\times m}$.

Remark 7.4 *If the ordering in* (7.2) *is chosen differently, the blocking in* (7.3) *is also different. In terms of bandwidth of the resulting matrix, the ordering given in* (7.2) *is far from optimal. However, the bandwidth of the matrix can be reduced very easily by available packages and algorithms after the assembling step. The given ordering makes the assembling rather simple, though and any other ordering would make this step much more complex.*

Elements of the Vector Valued Finite Element Space

At the beginning of this section, it was stated that all the factors of the vector valued FE space need to have the same mesh $\mathcal{T} = \{K\}$. This helps to define the elements[1] $(K, \underline{T}_K)$ of the vector valued space: they have the same support K as the scalar factors. What might be different from (scalar) space to space is the approximation degree, the boundary conditions and the number of local shape functions in that particular element. This is all condensed in the T matrix $\underline{T}_K^V$ of the (scalar) element $(K, \underline{T}_K^V)$ of the FE space V.

Therefore, the elements in the vector valued FE space $V \times W$ have the same support K and a different T matrix $\underline{T}_K$ than the elements in the scalar space. Looking at the assembly procedure of the blocks of the matrices in (7.3) reveals the relation of $\underline{T}_K^V$, $\underline{T}_K^W$ and $\underline{T}_K$.

Assembling of the Vector Valued Finite Element Space

By Definition 3.22, the blocks in (7.3) are assembled in the following way:

$$\underline{A}^{VV} = \sum_{K,\tilde{K}\in\mathcal{T}} (\underline{T}_K^V)^\top \underline{A}_{K\tilde{K}}^{VV} \underline{T}_{\tilde{K}}^V, \qquad \underline{A}^{VW} = \sum_{K,\tilde{K}\in\mathcal{T}} (\underline{T}_K^V)^\top \underline{A}_{K\tilde{K}}^{VW} \underline{T}_{\tilde{K}}^W,$$
$$\underline{A}^{WV} = \sum_{K,\tilde{K}\in\mathcal{T}} (\underline{T}_K^W)^\top \underline{A}_{K\tilde{K}}^{WV} \underline{T}_{\tilde{K}}^V, \qquad \underline{A}^{WW} = \sum_{K,\tilde{K}\in\mathcal{T}} (\underline{T}_K^W)^\top \underline{A}_{K\tilde{K}}^{WW} \underline{T}_{\tilde{K}}^W.$$

The element matrices $\underline{A}_{K\tilde{K}}^{VV}$, $\underline{A}_{K\tilde{K}}^{VW}$, $\underline{A}_{K\tilde{K}}^{WV}$ and $\underline{A}_{K\tilde{K}}^{WW}$ result from the evaluation of the bilinear form with the element combination $(K, \underline{T}_K)$–$(\tilde{K}, \underline{T}_{\tilde{K}})$ (elements from $V \times W$), just in the 'scalar' decomposition. Hence, the assembling in the vector

[1] In this section, we adopt the notation *(cell, T matrix)* for an element instead of *(cell, polynomial degree)* used elsewhere.

valued space:

$$\underline{A} = \sum_{K,\tilde{K}\in\mathcal{T}} \underline{T}_K^\top \underline{A}_{K\tilde{K}} \underline{T}_{\tilde{K}},$$

where

$$\underline{A}_{K\tilde{K}} := \begin{pmatrix} \underline{A}_{K\tilde{K}}^{VV} & \underline{A}_{K\tilde{K}}^{VW} \\ \underline{A}_{K\tilde{K}}^{WV} & \underline{A}_{K\tilde{K}}^{WW} \end{pmatrix} \text{ and } \underline{T}_K := \begin{pmatrix} \underline{T}_K^V & \\ & \underline{T}_K^W \end{pmatrix}. \tag{7.5}$$

Given $\underline{A}_{K\tilde{K}}^{VV} \in \mathbb{R}^{n_K \times n_{\tilde{K}}}$ and $\underline{A}_{K\tilde{K}}^{WW} \in \mathbb{R}^{m_K \times m_{\tilde{K}}}$, the size of $\underline{A}_{K\tilde{K}}$ is $(n_K + m_K) \times (n_{\tilde{K}} + m_{\tilde{K}})$. The T matrices have the sizes $T_K^V \in \mathbb{R}^{n_K \times n}$, $T_K^W \in \mathbb{R}^{m_K \times m}$ and $T_K \in \mathbb{R}^{(n_K+m_K)\times(n+m)}$.

With the element and T matrices in (7.5) and the basis of the FE space in (7.2), the requested properties 2 and 3 above are fulfilled (property 4 is also fulfilled but this is only shown in the next section).

7.1.2 Classes and Implementation

All the classes needed to implement the functionality described above are given in Figure 7.1. The classes have the same names as those for the scalar FE spaces introduced in Section 6.1.2. However, these two sets of classes reside in different namespaces. The basic classes are in the namespace concepts (upper part of Figure 7.1) and the classes for the vector valued Finite Elements in the namespace vectorial (lower part of Figure 7.1).

Almost all classes in the namespace vectorial depend on the class Vectorial[2]. This basic class is an abstract container for components in the vector valued Finite Elements setting. Vectorial is a parametrised type that can be specialised using a template argument. This template argument gives the type of the scalar component which should be vectorised. Vectorial takes care of the handling of the different components (addition, removal and storage of components).

Each of the classes TMatrix, LinearForm, BilinearForm, Element and Space in the namespace vectorial is a specialisation of Vectorial with the template parameter of Vectorial set accordingly (the label on the inheritance arrows in Figure 7.1 is the template argument comp).

[2] Upper / lower case is significant here!

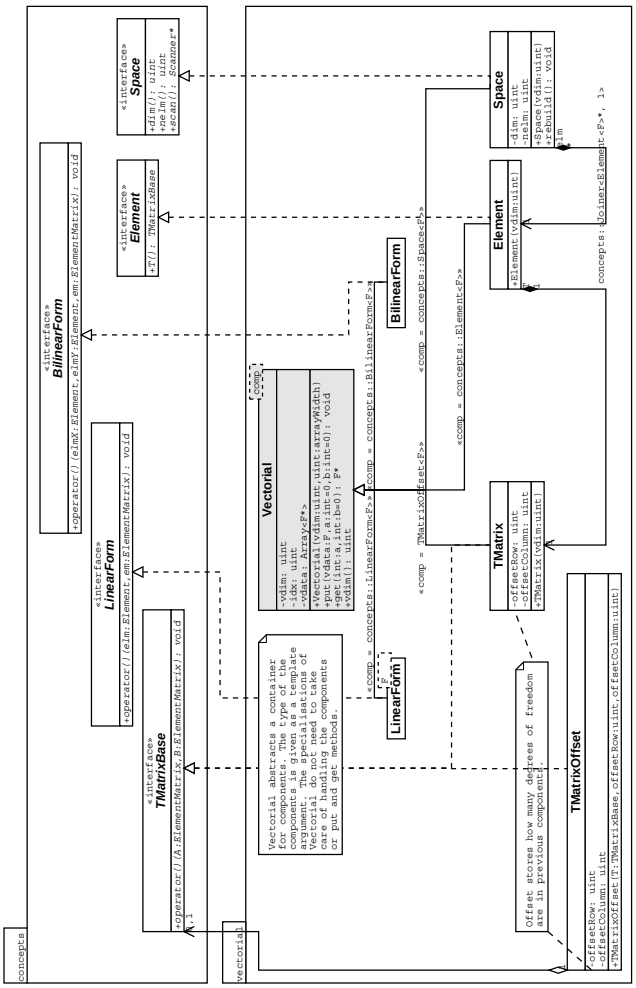


Figure 7.1: All classes in the namespace vectorial and their dependency on the main namespace concepts.

```
hp3D::Space scalarX(msh, 0, pX, bcX),
            scalarY(msh, 0, pY, bcY),
            scalarZ(msh, 0, pZ, bcZ);

vectorial::Space<Real> spc(3);
spc.put(scalarX);
spc.put(scalarY);
spc.put(scalarZ);
```

Algorithm 7.1: Creating a vector valued FE space.

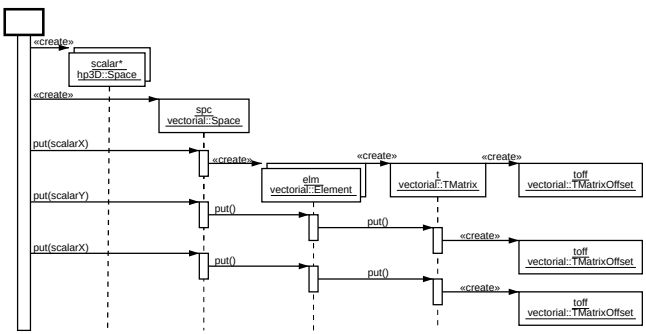


Figure 7.2: Sequence diagram for creation of a vector valued FE space. The left most life line symbolises the main program of the user.

Building a Vector Valued Finite Element Space

Building a vector valued FE space from the previously constructed scalar spaces is as simple as adding them with the put call to the vector valued space, *c. f.* Algorithm 7.1 and Figure 7.2. The first call to put generates the list of elements in the vector valued space. Every subsequent call to put adds the new elements as new components to the already generated elements in the vector valued FE space (using the put call of Element in the namespace vectorial).

An Element in vectorial receiving a put call from its space adds the element to its components and adds the T matrix of the newly arrived scalar element to

its T matrix (using a put call to TMatrix in the namespace vectorial). A TMatrix receiving a put call from an element stores the arriving T matrix as a component using a TMatrixOffset. TMatrix implements the idea of having different T matrices as blocks building a larger T matrix whereas TMatrixOffset implements the idea of having a T matrix with shifted indices (hence the name 'offset').

Bilinear and Linear Forms for Vector Valued Problems

In a similiar way to how a vector valued FE space was constructed, the vector valued bilinear and linear forms are built from scalar ones. The main difference to the construction of the space described before is that the user has to give an index to the put call. This index identifies the order in the block structure of the bilinear and linear forms. If a place in this block structure is not occupied by a scalar bilinear or linear form it is simply assumed to be identically zero. This makes it simple for the user to create 'sparse' block structures in the bilinear or linear form (by leaving out a block).

This implementation fulfils item 4 of the list of properties given at the beginning of Section 7.1.1: The user has to give the number of factors in the Cartesian product of the FE spaces. This number does not have an upper bound.

7.1.3 Example of a Bilinear Form

Writing the blocks of a bilinear form is straightforward after having seen the ideas once. They are presented using the example of the **curl–curl** operator from the discretisation of Maxwell's equations.

The **curl–curl** operator results in the bilinear form

$$a(\boldsymbol{u}, \boldsymbol{v}) = \int_K (\mathbf{curl}\, \boldsymbol{u})^\top \cdot \mathbf{curl}\, \boldsymbol{v}\, d\boldsymbol{x} = \int_K (\nabla_x \times \boldsymbol{u})^\top \cdot (\nabla_x \times \boldsymbol{v})\, d\boldsymbol{x}.$$

The integration is performed in the reference element $\hat{K}$ with the mapping $F_K : \hat{K} \to K, \boldsymbol{\xi} \mapsto \boldsymbol{x}$ and $\nabla_x = dF^{-\top} \cdot \nabla_\xi$. dF_K^{-1} is the inverse of the Jacobian matrix with the columns $\boldsymbol{a}_j$:

$$dF^{-1} = \left(\frac{\partial \xi_i}{\partial x_j} \right)_{i,j=1}^{3} = \begin{pmatrix} \boldsymbol{a}_1 & \boldsymbol{a}_2 & \boldsymbol{a}_3 \end{pmatrix}.$$

Thus with $\boldsymbol{N}(\boldsymbol{\xi}) := \boldsymbol{u} \circ F_K$ and $\boldsymbol{M}(\boldsymbol{\xi}) := \boldsymbol{v} \circ F_K$,

$$\begin{aligned} a(\boldsymbol{u}, \boldsymbol{v}) &= \int_{\hat{K}} \left((dF^{-\top} \cdot \nabla_{\xi}) \times \boldsymbol{N} \right)^{\top} \cdot \left((dF^{-\top} \cdot \nabla_{\xi}) \times \boldsymbol{M} \right) |dF| \, d\boldsymbol{\xi} \\ &= \int_{\hat{K}} \begin{pmatrix} \boldsymbol{a}_2^{\top} \nabla N_3 - \boldsymbol{a}_3^{\top} \nabla N_2 \\ \boldsymbol{a}_3^{\top} \nabla N_1 - \boldsymbol{a}_1^{\top} \nabla N_3 \\ \boldsymbol{a}_1^{\top} \nabla N_2 - \boldsymbol{a}_2^{\top} \nabla N_1 \end{pmatrix}^{\top} \begin{pmatrix} \boldsymbol{a}_2^{\top} \nabla M_3 - \boldsymbol{a}_3^{\top} \nabla M_2 \\ \boldsymbol{a}_3^{\top} \nabla M_1 - \boldsymbol{a}_1^{\top} \nabla M_3 \\ \boldsymbol{a}_1^{\top} \nabla M_2 - \boldsymbol{a}_2^{\top} \nabla M_1 \end{pmatrix} |dF| \, d\boldsymbol{\xi} \\ &= \sum_{i=1}^{3} \sum_{j=1}^{3} \int_{\hat{K}} \nabla N_i^{\top} \underline{M}_{ij} \nabla M_j \, |dF| \, d\boldsymbol{\xi}, \end{aligned} \tag{7.6}$$

where $\underline{M}_{ij}$ is a 3×3 matrix according to the following table ($\underline{M}_{ij} = \underline{M}_{ji}^{\top}$):

$i \setminus j$	1	2	3
1	$\boldsymbol{a}_2\boldsymbol{a}_2^{\top} + \boldsymbol{a}_3\boldsymbol{a}_3^{\top}$	$-\boldsymbol{a}_2\boldsymbol{a}_1^{\top}$	$-\boldsymbol{a}_3\boldsymbol{a}_1^{\top}$
2	$-\boldsymbol{a}_1\boldsymbol{a}_2^{\top}$	$\boldsymbol{a}_1\boldsymbol{a}_1^{\top} + \boldsymbol{a}_3\boldsymbol{a}_3^{\top}$	$-\boldsymbol{a}_3\boldsymbol{a}_2^{\top}$
3	$-\boldsymbol{a}_1\boldsymbol{a}_3^{\top}$	$-\boldsymbol{a}_3\boldsymbol{a}_1^{\top}$	$\boldsymbol{a}_1\boldsymbol{a}_1^{\top} + \boldsymbol{a}_2\boldsymbol{a}_2^{\top}$.

As shown in (7.3), the bilinear form has a block structure. In our case, $a(.\,,.)$ has a 3×3 block structure:

$$a(\boldsymbol{u}, \boldsymbol{v}) = \begin{pmatrix} a_{11}(u_1, v_1) & a_{12}(u_1, v_2) & a_{13}(u_1, v_3) \\ a_{21}(u_2, v_1) & a_{22}(u_2, v_2) & a_{23}(u_2, v_3) \\ a_{31}(u_3, v_1) & a_{32}(u_3, v_2) & a_{33}(u_3, v_3) \end{pmatrix}. \tag{7.7}$$

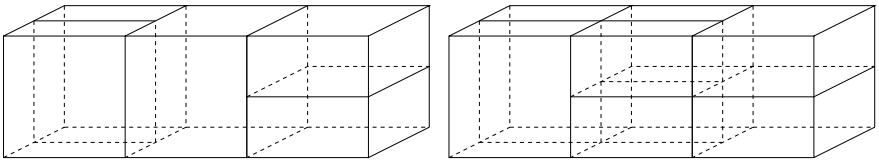
Therefore, the sum in (7.6) degenerates to

$$a_{ij}(u_i, v_j) = \int_{\hat{K}} \nabla N_i^{\top} \underline{M}_{ij} \nabla M_j \, |dF| \, d\boldsymbol{\xi}$$

in one of the blocks in (7.7). The coefficient's matrix $\underline{M}_{ij}$ is suitable for the sum factorisation framework presented in Section 6.4.

7.2 Geometric Deadlock Problem

This problem has already been mentioned in Section 6.2.2.



(a) Refinement in the left and right hexahedra are given.

(b) Compatible refinement.

Figure 7.3: Deadlock problem.

The mesh in Figure 7.3(a) shows a deadlock situation. The middle hexahedron cannot be refined in any of the ways the left or right hexahedron were refined. The problem arises at the shared faces: A shared face would have to be refined in two incompatible ways.

Remark 7.8 *This situation does normally not occur when refining a mesh by hand or using an a-priori algorithm like Algorithm 3.3. However, in an automatic hp-adaptive algorithm, this could happen.*

The mesh in Figure 7.3(b) shows a possible solution to the above mentioned deadlock problem: The middle hexahedron is refined into four children. This way, the shared faces can be refined in a compatible way. Namely, each of the two children of the first refinement of the faces (given in Figure 7.3(a)) are refined themselves into two children. These small quadrilaterals can then be used to define the four new hexahedra as children of the middle hexahedron.

The subdivision algorithm automatically upgrades an incompatible refinement to a compatible one. This is done by a 'trial and error' method: As long as the requested refinement is incompatible (reported by an exception *c. f.* Section 6.2.2), it is upgraded in an appropriate way and retried. If even a subdivision into eight children fails, it is discarded completely (with a message to the user).

The algorithms in Section 6.3 need some adaptions to the new situation since one main implicit precondition has changed: The faces of a cell are no longer refined in the same way as the cell itself: If a geometric deadlock situation has to be solved, a face is refined into four children in two steps and the cell itself is refined into four children in one step. To reflect this, some technical special cases have to be dealt with in the algorithms. However, the basic ideas stay the same.

```
timeval* utime = &ru_.ru_utime;
timeval* stime = &ru_.ru_stime;
float stamp = (utime->tv_sec * 1.0e6 + utime->tv_usec +
               stime->tv_sec * 1.0e6 + stime->tv_usec) / 1.0e6;

getrusage(RUSAGE_SELF, &ru_);

float ut = (utime->tv_sec * 1.0e6 + utime->tv_usec) / 1.0e6;
float st = (stime->tv_sec * 1.0e6 + stime->tv_usec) / 1.0e6;

float res = ut+st-stamp;
```

Algorithm 7.2: Run-time measurements. ru is a persistent variable of this routine. stamp $= ts_{i-1}$ and res $= \Delta t_i$.

7.3 CPU and Timing Information

7.3.1 Hardware and Compiler

The computations were done on a Sun Fire 880 with 32 GBytes of physical main memory and 8 processors at 800 MHz—only using one processor, though: Concepts is neither using parallelism with threads (shared memory) nor message passing (MPI, distributed memory). GCC 3 [57] was used to compile the software.

7.3.2 Code for Run-Time Measurements

The getrusage method of the C library returns a struct containing (amongst other things) timing information in the ru_utime (user time used ut) and ru_stime (system time used st) fields. We compute the used time Δt_i by the difference between two time stamps $ts_i = ut_i + st_i$: $\Delta t_i = ts_{i+1} - ts_i$ (*c. f.* Algorithm 7.2). Note that the minimal resolution of the timer is 0.01 s.

7.4 History and Authors of Concepts

The software was mainly written by Dr. Christian Lage. The first versions and many of the ideas behind the software appear in his Ph.D. thesis [73] in the early 90ies of the last century. The predecessor of the current version was developed during his post doctoral studies at the Seminar for Applied Mathematics of the Swiss Federal Institute of Technology (ETH), Zurich. The design ideas leading to this version are summarised in [74].

Dr. Ana-Maria Matache worked on the hp-FEM part of Concepts. She implemented the quadrilateral elements for two dimensional problems together with Christian during her Ph.D. studies [82]. It was her who gave the author his first introductions to Concepts.

Dr. Gregor Schmidlin worked on the BEM part of Concepts during his Ph.D. studies [93].

Many students have also worked on and with Concepts:

- In summer 1999, David Hoch and Andreas Rüegg implemented mixed and variable boundary conditions for hp-FEM as a term project.

- During the winter term 1999/2000, the author wrote his diploma thesis based on Concepts. At this time, he implemented hp-DGFEM for a second order PDE with constant coefficients and mixed and variable boundary conditions.

- During the winter term 2000/2001, Andreas Rüegg wrote his diploma thesis about generalised FEM. He continues to work with generalised FEM at our institute as a Ph.D. student.

- In the summer term 2002, Manuel Walser designed and implemented time stepping schemes (in particular the Newmark scheme) for discretised problems as a term project. Norbert Fernandes designed and implemented an interface to the Eigenvalue solver JDBSYM [58, 59].

- In the winter term 2002/2003, Christoph Winkelmann and Adrian Burri used Concepts to solve non-linear convection-diffusion problems using a Finite Volume Discontinuous Galerkin Scheme. During their work, the time-stepping classes were improved considerably.

- During the winter term 2003/2004, Christoph Winkelmann wrote his diploma thesis using Concepts on DGFEM in two dimensions for time dependent saddle point problems.

After completing his diploma thesis, the author started to work on three dimensional *hp*-FEM in Concepts as part of his Ph.D. studies in spring 2000.

In spring 2002, Kersten Schmidt began to work on vector valued problems and implemented the namespace vectorial and the necessary bilinear forms required to solve Maxwell's equations using weighted regularisation. He is now implementing *hp* edge elements for quadrilaterals and hexahedra for his Ph.D. research.

Open Source

The author has been successful in convincing all contributors and his advisor Prof. Christoph Schwab that an open source [90] version of Concepts should be released [26]. We chose the GNU General Public License (GPL) [53] as the license for the public part of Concepts. Some parts (where active research is still going on) remain closed to the public but might be released under the same license at a later date.

Bibliography

[1] Robert A. Adams. *Sobolev spaces*. Academic Press, New York, London, 1975. Pure and Applied Mathematics, Vol. 65.

[2] Mark Ainsworth. *Essential boundary conditions and multi-point constraints in finite element analysis*. Comput. Methods Appl. Mech. Engrg., 190(48):6323–6339, doi:10.1016/S0045-7825(01)00236-5, 2001.

[3] Mark Ainsworth and Joe Coyle. *Hierarchic hp-edge element families for Maxwell's equations on hybrid quadrilateral/triangular meshes*. Comput. Methods Appl. Mech. Engrg., 190(49–50):6709–6733, doi:10.1016/S0045-7825(01)00259-6, 2001.

[4] Mark Ainsworth, Joe Coyle, Paul D. Ledger, and Ken Morgan. *Computing Maxwell Eigenvalues by Using Higher Order Edge Elements in Three Dimensions*. IEEE Transactions on Magnetics, 39(5):2149–2153, doi:10.1109/TMAG.2003.817097, September 2003.

[5] Mark Ainsworth and Katia Pinchedez. *hp-approximation theory for BDFM and RT finite elements on quadrilaterals*. SIAM J. Numer. Anal., 40(6):2047–2068, doi:10.1137/S0036142901391128, 2002.

[6] Jochen Alberty, Carsten Carstensen, and Stefan A. Funken. *Remarks around 50 lines of Matlab: short finite element implementation*. Numer. Algorithms, 20(2–3):117–137, 1999.

[7] *Asgard Beowulf Cluster*, `http://www.asgard.ethz.ch/`.

[8] Ivo Babuška and Benqi Guo. *Approximation properties of the h-p version of the finite element method*. Comput. Methods Appl. Mech. Engrg., 133(3-4):319–346, 1996.

[9] Ivo Babuška and John E. Osborn. *Finite Element Methods (Part I)*, volume II of *Handbook of numerical analysis*, chapter Eigenvalue problems, pages 641–787. North-Holland, 1991.

[10] Ivo Babuška and Manil Suri. *The p and h-p versions of the finite element method, basic principles and properties*. SIAM Rev., 36(4):578–632, 1994.

[11] Satish Balay, Kris Buschelman, William D. Gropp, Dinesh Kaushik, Lois Curfman McInnes, and Barry F. Smith, *PETSc home page*, `http://www.mcs.anl.gov/petsc`, 2001.

[12] Satish Balay, William D. Gropp, Lois Curfman McInnes, and Barry F. Smith. *Efficient Management of Parallelism in Object Oriented Numerical Software Libraries*. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhauser Press, 1997.

[13] Satish Balay, William D. Gropp, Lois Curfman McInnes, and Barry F. Smith. *PETSc Users Manual*. Technical Report ANL-95/11 – Revision 2.1.0, Argonne National Laboratory, 2001.

[14] Wolfgang Bangerth, Ralf Hartmann, and Guido Kanschat. `deal.II` *Differential Equations Analysis Library, Technical Reference*. IWR Heidelberg.

[15] Wolfgang Bangerth and Guido Kanschat. *Concepts for Object-Oriented Finite Element Software – the* `deal.II` *library*. Preprint 99-43 (SFB 359), IWR Heidelberg, 10 1999.

[16] Rick Beatson and Leslie Greengard. *A short course on fast multipole methods*. In Mark Ainsworth and J. Levesley, editors, *Wavelets, multilevel methods and elliptic PDEs*, Numer. Math. Sci. Comput., pages 1–37. Oxford Univ. Press, New York, 1997.

[17] Richard E. Bellman, editor. *Stochastic Processes in Mathematical Physics and Engineering*, volume 16 of *Proceedings of Symposia in Applied Mathematics*. American Mathematical Society, 1964.

[18] *Beowulf Cluster Site*, `http://www.beowulf.org/`.

[19] Susanne C. Brenner and L. Ridgway Scott. *The mathematical theory of finite element methods*, volume 15 of *Texts in Applied Mathematics*. Springer-Verlag, New York, 1994.

[20] Franco Brezzi and Michel Fortin. *Mixed and hybrid finite element methods*, volume 15 of *Springer Series in Computational Mathematics*. Springer-Verlag, New York, 1991.

[21] Timothy A. Budd. *An introduction to object oriented programming*. Addison-Wesley, 1991.

[22] Françoise Chatelin. *Eigenvalues of matrices*. John Wiley & Sons Ltd., Chichester, 1993. Translated from the French and with additional material by Walter Ledermann.

[23] Noam Chomsky. *Syntactic Structures*, volume 4 of *Janua Linguarum Series Minor*. 'S-Gravenhage Mouton, 1963.

[24] Philippe G. Ciarlet. *The finite element method for elliptic problems*. North-Holland Publishing Co., Amsterdam, 1978. Studies in Mathematics and its Applications, Vol. 4.

[25] Philippe G. Ciarlet. *Finite Element Methods (Part I)*, volume II of *Handbook of numerical analysis*, chapter Basic Error Estimates for Elliptic Problems, pages 17–351. North-Holland, 1991.

[26] Concepts Development Team, *Concepts Homepage*, `http://www.concepts.math.ethz.ch/`, 2003.

[27] Martin Costabel. *A coercive bilinear form for Maxwell's equations*. J. Math. Anal. Appl., 157(2):527–541, 1991.

[28] Martin Costabel and Monique Dauge. *Maxwell and Lamé eigenvalues on polyhedra*. Math. Methods Appl. Sci., 22(3):243–258, 1999.

[29] Martin Costabel and Monique Dauge. *Weighted regularization of Maxwell equations in polyhedral domains. A rehabilitation of nodal finite elements*. Numer. Math., 93(2):239–277, doi:10.1007/s002110100388, 2002.

[30] Martin Costabel, Monique Dauge, and Serge Nicaise. *Singularities of Maxwell interface problems*. M2AN Math. Model. Numer. Anal., 33(3):627–649, 1999.

[31] Martin Costabel, Monique Dauge, and Christoph Schwab. *Exponential Convergence of hp-FEM for Maxwell's Equations with Weighted Regularization in Polygonal Domains*. Technical Report 2004-05, Seminar for Applied Mathematics, ETH Zurich, 2004. Submitted to M3AS.

[32] Martin Costabel, Monique Dauge, and Christoph Schwab. *Exponential Convergence of hp-FEM for Maxwell's Equations with Weighted Regularization*. In preparation, 2005.

[33] Joe Coyle and Paul D. Ledger. *Evidence of exponential convergence in the computation of Maxwell eigenvalues*. To appear in Comput. Methods Appl. Mech. Engrg., 2003.

[34] Eric Darve. *The fast multipole method. I. Error analysis and asymptotic complexity*. SIAM J. Numer. Anal., 38(1):98–128 (electronic), doi:10.1137/S0036142999330379, 2000.

[35] Eric Darve. *The fast multipole method: numerical implementation*. J. Comput. Phys., 160(1):195–240, doi:10.1006/jcph.2000.6451, 2000.

[36] Monique Dauge, *Benchmark computations for Maxwell equations for the approximation of highly singular solutions*, `http://perso.univ-rennes1.fr/monique.dauge/benchmax.html`, 2002.

[37] Monique Dauge, *Private Communication*, April 2004.

[38] Tim A. Davis. *A column pre-ordering strategy for the unsymmetric-pattern multifrontal method*. Technical Report TR-03-006, Departement of Computer and Information Science and Engineering, University of Florida, 2003. Submitted to ACM Trans. Math. Software.

[39] Tim A. Davis, *UMFPACK Homepage*, `http://www.cise.ufl.edu/research/sparse/umfpack/`, 2003.

[40] Tim A. Davis. *UMFPACK Version 4.1 User Guide*. Technical Report TR-03-008, Univ. of Florida, CISE Dept., Gainesville, FL, 2003.

[41] Leszek Demkowicz, J. Tinsely Oden, Waldemar Rachowicz, and O. Hardy. *Toward a universal h-p adaptive finite element strategy. I. Constrained approximation and data structure*. Comput. Methods Appl. Mech. Engrg., 77(1-2):79–112, doi:10.1016/0045-7825(89)90129-1, 1989.

[42] Leszek Demkowicz, David Pardo, and Waldemar Rachowicz, *Homepage for 3Dhp90*, `http://www.ticam.utexas.edu/~leszek/3Dhp90.html`.

[43] Leszek Demkowicz, David Pardo, and Waldemar Rachowicz. *3D hp-Adaptive Finite Element Package (3Dhp90). Version 2.0, The Ultimate Data Structure for Three Dimensional, Anisotropic hp Refinements*. Technical Report 02-24, TICAM, 2002.

[44] Leszek Demkowicz, Waldemar Rachowicz, and Philippe R. B. Devloo. *A fully automatic hp-adaptivity*. In *Proceedings of the Fifth International Conference on Spectral and High Order Methods (ICOSAHOM-01) (Uppsala)*, volume 17, pages 117–142, doi:10.1023/A:1015192312705, 2002.

[45] James W. Demmel, John R. Gilbert, and Xiaoye S. Li, *SuperLU Homepage*, `http://www.nersc.gov/~xiaoye/SuperLU/`, 1999.

[46] James W. Demmel, John R. Gilbert, and Xiaoye S. Li. *SuperLU Users' Guide*. Technical Report LBNL-44289, Lawrence Berkely National Lab, 2003.

[47] Philippe R. B. Devloo. *PZ: An object oriented environment for scientific programming*. Comput. Methods Appl. Mech. Engrg., 150(1–4):133–153, doi:10.1016/S0045-7825(97)00097-2, 1997.

[48] Alexander Düster. *High order finite elemnts for three-dimensional, thin-walled nonlinear continua*. PhD thesis, Technische Universität München, 2001.

[49] Alexander Düster, Henrike Bröker, and Ernst Rank. *The p-version of the Finite Element Method for three-dimensional curved thin walled structures*. Int. J. Numer. Methods Engng, 52(7):673–703, doi:10.1002/nme.222, 2001.

[50] Philippe R. B. Devloo et. al., *PZ Homepage*, `http://labmec.fec.unicamp.br/~pz/`, 2002.

[51] Lawrence C. Evans. *Partial differential equations*, volume 19 of *Graduate Studies in Mathematics*. American Mathematical Society, Providence, RI, 1998.

[52] Miroslav Fiedler. *Special matrices and their applications in numerical mathematics*, chapter Tensor Product of Matrices. Compound Matrices, page 136ff. Martinus Nijhoff Publishers, 1986. Translated from the Czech by Petr Přikryl and Karel Segeth.

[53] Free Software Foundation, *The GNU General Public License (GPL)*, `http://www.opensource.org/licenses/gpl-license.php`, June 1991.

[54] Philipp Frauenfelder and Christian Lage. *Concepts—An Object-Oriented Software Package for Partial Differential Equations*. M2AN Math. Model. Numer. Anal., 36(5):937–951, doi:10.1051/m2an:2002036, 2002.

[55] Philipp Frauenfelder, Christoph Schwab, and Radu A. Todor. *Finite elements for elliptic problems with stochastic coefficients*. Comput. Methods Appl. Mech. Engrg., doi:10.1016/j.cma.2004.04.008, 2004. To appear.

[56] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Professional Computing Series. Addison-Wesley, 1995.

[57] *GCC—The GNU Compiler Collection*, `http://www.gnu.org/software/gcc/`, 2003.

[58] Roman Geus. *The Jacobi-Davidson algorithm for solving large sparse symmetric eigenvalue problems with application to the design of accelerator cavities*. PhD thesis14734, ETH Zurich, 2002.

[59] Roman Geus and Oscar Chinellato, *JDBSYM 0.14*, `http://www.inf.ethz.ch/personal/geus/software.html`, 2000.

[60] Roger G. Ghanem and Pol D. Spanos. *Stochastic finite elements: a spectral approach*. Springer-Verlag, New York, 1991.

[61] Vivette Girault and Pierre-Arnaud Raviart. *Finite element methods for Navier-Stokes equations*, volume 5 of *Springer Series in Computational Mathematics*. Springer-Verlag, Berlin, 1986. Theory and algorithms.

[62] Gene H. Golub. *Some modified matrix eigenvalue problems*. SIAM Rev., 15:318–334, 1973.

[63] Wen Z. Gui and Ivo Babuška. *The h, p and h-p versions of the finite element method in* 1 *dimension. I: The error analysis of the p-version, II: The error analysis of the h- and hp-version, III: The adaptive h-p version*. Numer. Math., 49(6):577–683, 1986.

[64] Wolfgang Hackbusch. *Integral equations*, volume 120 of *International Series of Numerical Mathematics*. Birkhäuser Verlag, Basel, 1995. Theory and numerical treatment, Translated and revised by the author from the 1989 German original.

[65] Wolfgang Hackbusch and Zenon P. Nowak. *On the fast matrix multiplication in the boundary element method by panel clustering*. Numer. Math., 54(4):463–491, 1989.

[66] J. M. Hammersley and D. C. Handscomb. *Monte Carlo methods*. Methuen & Co. Ltd., London, 1965.

[67] Christophe Hazard and Marc Lenoir. *On the solution of time-harmonic scattering problems for Maxwell's equations*. SIAM J. Math. Anal., 27(6):1597–1630, 1996.

[68] Vincent Heuveline, *HiFlow Homepage*, `http://www.hiflow.de/`, 2002.

[69] Engineering Software Research & Development Inc., *StressCheck Homepage*, `http://www.esrd.com/`, 2003.

[70] John David Jackson. *Classical electrodynamics*. John Wiley & Sons Inc., New York, second edition, 1975.

[71] Malvin H. Kalos and Paula A. Whitlock. *Monte Carlo methods. Vol. I*. A Wiley-Interscience Publication. John Wiley & Sons Inc., New York, 1986. Basics.

[72] George Em Karniadakis and Spencer J. Sherwin. *Spectral/hp element methods for CFD*. Numerical Mathematics and Scientific Computation. Oxford University Press, New York, 1999.

[73] Christian Lage. *Softwareentwicklung zur Randelementmethode: Analyse und Entwurf effizienter Techniken*. PhD thesis, Christian-Albrechts-Universität, Kiel, 1995.

[74] Christian Lage. *Concept Oriented Design of Numerical Software*. Technical Report 98-07, Seminar for Applied Mathematics, ETH Zurich, 7 1998.

[75] *LAPACK Homepage*, `http://www.netlib.org/lapack/`.

[76] Paul D. Ledger. *An hp-adaptive finite element procedure for electromagnetic scattering problems*. PhD thesis, University of Wales, November 2001.

[77] Richard B. Lehoucq, Kristyn J. Maschhoff, Danny C. Sorensen, and Chao Yang, *ARPACK Homepage*, `http://www.caam.rice.edu/software/ARPACK/`.

[78] Richard B. Lehoucq, Danny C. Sorensen, and Chao Yang. *ARPACK users' guide*. Software, Environments, and Tools. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1998. Solution of large-scale eigenvalue problems with implicitly restarted Arnoldi methods.

[79] Rolf Leis. *Zur Theorie elektromagnetischer Schwingungen in anisotropen inhomogenen Medien*. Math. Z., 106:213–224, 1968.

[80] Michel Loève. *Probability theory. I*, volume 45 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, fourth edition, 1977.

[81] Michel Loève. *Probability theory. II*, volume 46 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, fourth edition, 1978.

[82] Ana-Maria Matache. *Spectral and p-Finite Elements for problems with microstructure*. PhD thesis 13815, ETH Zurich, 2001.

[83] Jens M. Melenk. *hp-Finite Element Methods for Singular Perturbations*, volume 1796 of *Lecture Notes in Mathematics*. Springer-Verlag, Berlin, 2002.

[84] Jens M. Melenk, Klaus Gerdes, and Christoph Schwab. *Fully Discrete hp-Finite Elements: Fast Quadrature*. Comput. Methods Appl. Mech. Engrg., 190(32–33):4339–4364, doi:10.1016/S0045-7825(00)00322-4, 2001.

[85] Jens M. Melenk and Christoph Schwab. *H P FEM for reaction-diffusion equations. I. Robust exponential convergence*. SIAM J. Numer. Anal., 35(4):1520–1557 (electronic), doi:10.1137/S0036142997317602, 1998.

[86] David Musser and Alexander A. Stepanov. *Generic Programming*. In *ISSAC: Proceedings of the ACM SIGSAM International Symposium on Symbolic and Algebraic Computation*, 1989.

[87] Jean-Claude Nédélec. *Mixed finite elements in* $\mathbb{R}^3$. Numer. Math., 35(3):315–341, 1980.

[88] Jean-Claude Nédélec. *A new family of mixed finite elements in* $\mathbb{R}^3$. Numer. Math., 50(1):57–81, 1986.

[89] Object Management Group, Inc., Framingham, USA. *OMG Unified Modeling Language Specification*, 1999.

[90] Bruce Perens, *The Open Source Definition*, `http://www.opensource.org/docs/definition.php`, 2004. The first draft of this document was released as "The Debian Free Software Guidelines" in 1997. The Debian-specific references were removed from the document to create the "Open Source Definition".

[91] Olaf Schenk, Klaus Gärtner, Wolfgang Fichtner, and Andreas Stricker. *PARDISO: a high-performance serial and parallel sparse linear solver in semiconductor device simulation*. Future Generation Computer Systems, 18(1):69–78, doi:10.1016/S0167-739X(00)00076-5, 2001.

[92] Martin Schlather. *Introduction to positive definite functions and to unconditional simulation of random fields*. Technical Report ST-99-10, Department of Mathematics and Statistics, Lancaster University, 1999.

[93] Gregor Schmidlin. *Fast Solution Algorithms for Integral Equations in* $\mathbb{R}^3$. PhD thesis 15016, ETH Zurich, 2003.

[94] Gregor Schmidlin, Christian Lage, and Christoph Schwab. *Rapid solution of first kind boundary integral equations in* $\mathbb{R}^3$. Engineering Analysis with Boundary Elements, 27(5):469–490, doi:10.1016/S0955-7997(02)00156-X, 2003.

[95] Joachim Schöberl, *Private Communication*, November 2003.

[96] Dominik Schötzau, Christoph Schwab, and Andrea Toselli. *Stabilized hp-DGFEM for Incompressible Flow*. Math. Models Meth. Appl. Sci., 13(10):1413–1436, doi:10.1142/S0218202503002970, 2003.

[97] Christoph Schwab. *p- and hp-finite element methods*. Numerical Mathematics and Scientific Computation. The Clarendon Press Oxford University Press, New York, 1998. Theory and applications in solid and fluid mechanics.

[98] Christoph Schwab and Manil Suri. *The p and hp versions of the finite element method for problems with boundary layers*. Math. Comp., 65(216):1403–1429, 1996.

[99] Christoph Schwab and Radu A. Todor. *Approximation of Random Fields using generalised Fast Multipole Methods*. In preparation.

[100] Christoph Schwab and Radu A. Todor. *Sparse finite elements for elliptic problems with stochastic loading*. Numer. Math., 95(4):707–734, doi:10.1007/s00211-003-0455-z, 2003.

[101] Jeremy G. Siek and Andrew Lumsdaine. *The Matrix Template Library: Generic Components for High-Performance Scientific Computing*. Computing in Science & Engineering, 1(6):70–78, November/December 1999.

[102] Alexander A. Stepanov and Meng Lee. *The Standard Template Library*. Hewlett-Packard Laboratories, Palo Alto, CA, October 1995.

[103] Gilbert Strang and George J. Fix. *An analysis of the finite element method*. Prentice-Hall Inc., Englewood Cliffs, N. J., 1973. Prentice-Hall Series in Automatic Computation.

[104] Bjarne Stroustrup. *A history of C++: 1979–1991*. In *The second ACM SIGPLAN conference on History of programming languages*, volume 28 of *ACM SIGPLAN Notices*, pages 271–297. SIGPLAN, ACM Press, New York, doi:10.1145/154766.155375, March 1993.

[105] Bjarne Stroustrup. *The C++ Programming Language*. Addison Wesley Longman, Inc., third edition, 1997.

[106] Barna Szabó and Ivo Babuška. *Finite element analysis*. A Wiley-Interscience Publication. John Wiley & Sons Inc., New York, 1991.

[107] Radu A. Todor. *Numerical treatment of stochastic PDEs*. PhD thesis, Swiss Fedral Institute of Technology Zurich. In preparation.

[108] Andrea Toselli. *hp discontinuous Galerkin approximations for the Stokes problem*. Math. Models Methods Appl. Sci., 12(11):1565–1597, doi:10.1142/S0218202502002240, 2002.

[109] Andrea Toselli and Christoph Schwab. *Mixed hp-finite element approximations on geometric edge and boundary layer meshes in three dimensions*. Numer. Math., 94(4):771–801, 2003.

[110] N. Wiener. *The homogeneous chaos*. Amer. J. Math., 60:897–936, 1930.

[111] N. Wiener. *Nonlinear Problems in Random Theory*. MIT Press and John Wiley and Sons, New York, 1958.

[112] Thomas P. Wihler. *Discontinuous Galerkin FEM for Elliptic Problems in Polygonal Domains*. PhD thesis 14973, ETH Zurich, 2002.

[113] Dongbin Xiu and George Em Karniadakis. *The Wiener-Askey polynomial chaos for stochastic differential equations*. SIAM J. Sci. Comput., 24(2):619–644, doi:10.1137/S1064827501387826, 2002.

Curriculum Vitae

On September 19, 1974, I was born in Winterthur, Switzerland. I visited the mandatory schools in Niederglatt (Kanton Zürich) and the Kantonsschule Zürcher Unterland in Bülach. In January 1994, I received the Matura Typus C.

In fall 1994, I matriculated at ETH Zürich to study mathematics. I graduated in spring 2000 with degree Dipl. Math. ETH.

Since April 2000, I have been working as a research and teaching assistant at the Seminar for Applied Mathematics at ETH Zürich. During this time, I developed my dissertation thesis in the field of numerical mathematics supervised by Prof. Dr. Christoph Schwab.

Since October 2004, I am working as a consultant for Solution Providers AG, Dübendorf.