# An Adaptive Wavelet Method for the Solution of Boundary Integral Equations in Three Dimensions

**Inauguraldissertation**

zur
Erlangung der Würde eines Doktors der Philosophie

vorgelegt der
Philosophisch-Naturwissenschaftlichen Fakultät
der Universität Basel

von
Manuela Utzinger

aus
Frenkendorf, Basel-Landschaft

Basel, 2016

Genehmigt von der Philosophisch-Naturwissenschaftlichen Fakultät
auf Antrag von

Prof. Dr. Helmut Harbrecht
Prof. Dr. Angela Kunoth

Basel, den 15.11.2016

Prof. Dr. Jörg Schibler
Dekan

# Acknowledgements

I want to use this opportunity to express my gratitude towards several important people, which have been by my side during my time as a Ph.D. student.

First and foremost, I want to thank my supervisor Prof. Dr. Helmut Harbrecht for giving me the opportunity to do a Ph.D. as a member of his research group. I am very grateful for his constant support, for the fascinating subject I was assigned to work on, and for all the fruitful discussions we have engaged in. The time in his group made it possible for me to acquire valuable skills and helped me to turn curiosity into insight. It was a great experience and I will fondly look back on the days spent here. Furthermore, I would like to kindly thank Prof. Dr. Angela Kunoth for agreeing to be the co-referee in my thesis committee. I know this task is time consuming and I appreciate the fact that she has taken the time and effort to evaluate my work.

I also would like to thank the other, current and former, members of our research group, for being available for mathematical, as well as trivial, discussions, for having lunch together, for drinking tons of coffee with me, and also for spending time together outside of work. You have all been like a family to me and I will always remember the awesome time in this group.

Life in the department would have been much less memorable, hadn't it been for so many nice people that touched my life in so many ways. Special thanks go to Barbara Fridez, Deborah Scheiblechner and Heidi Karypidis for having organised many morning and afternoon coffee breaks, helping me whenever I needed an auditorium or access to parts of the website and much more. You form a core part of this department and chaos would break loose if it weren't for you.

Many thanks go to Monica Bugeanu and Marie Kray for reading parts of this thesis and giving me many useful suggestions on how to improve it. My sincerest thanks go to Jürgen Dölz, which has taken the time to proofread my thesis, even during very stressful phases of his own work.

I am deeply grateful for the support of an indispensable group of fellow female mathematicians with whom it was always a pleasure to attend social events. Furthermore, I want to thank all my other non-mathematician friends for being in my life, for spending time with me when I needed a break from work and for cheering me up when life itself seemed overwhelming.

Last but not least, I want to thank my partner Michael Moor for his love and support, especially during the last months of my Ph.D. I love you more than words can say and I hope you will be by my side for many years to come.

# Contents

**Introduction** **1**

**1 Boundary Element Method** **9**

1.1 Laplace Equation . . . 11

1.1.1 Dirichlet Problem for the Laplace Equation . . . 11

1.1.2 Neumann Problem for the Laplace Equation . . . 13

1.2 Helmholtz Equation . . . 13

1.2.1 Sommerfeld's Radiation Condition . . . 14

1.2.2 Reformulation as a Boundary Integral Equation . . . 16

1.2.3 Scattering Problems . . . 18

1.3 Variational Formulation and Galerkin Scheme . . . 19

1.4 Fast Boundary Element Methods . . . 21

**2 Wavelets** **25**

2.1 Primal and Dual Scaling Functions on $\mathbb{R}$ . . . 26

2.2 Wavelet Bases on $\mathbb{R}$ . . . 27

2.3 Wavelet Bases on the Interval . . . 29

2.3.1 Scaling Functions . . . 29

2.3.2 Haar Basis . . . 30

2.3.3 Wavelets With Three Vanishing Moments . . . 31

2.4 Wavelet Bases on the Unit Square . . . 33

2.4.1 Ordinary Tensor Product Wavelets . . . 34

2.4.2 Wavelets With Optimised Support . . . 34

2.5 Wavelet Bases on the Surface . . . 35

**3 Adaptive Wavelet Schemes** **39**

3.1 Motivation and Background . . . 39
3.2 Nonlinear Approximation . . . 41
3.2.1 Best $N$-Term Approximation . . . 41
3.2.2 Approximation Spaces . . . 42
3.2.3 Sequence Spaces, Weak $\ell^p$ Spaces and Interpolation Spaces . . . 43
3.2.4 Besov Spaces . . . 44
3.3 On Adaptive Wavelet Algorithms . . . 49
3.3.1 Trees . . . 50
3.3.2 Coarse . . . 52
3.3.3 Apply . . . 54
3.3.4 Rhs . . . 58
3.3.5 Solve . . . 58

**4 Implementation** **59**

4.1 Element Trees . . . 60
4.1.1 Data Structures . . . 60
4.1.2 Initialisation of the Element Tree . . . 61
4.1.3 Element Refinement . . . 64
4.2 Wavelet Trees . . . 66
4.2.1 Initialisation of the Wavelet Tree . . . 67
4.2.2 Wavelet Archive . . . 67
4.2.3 Wavelet Refinement and Inheritance . . . 70
4.2.4 Wavelets with Three Vanishing Moments . . . 72
4.2.5 Switching Between Wavelet Bases . . . 73
4.3 The Adaptive Algorithm . . . 77
4.3.1 Overview . . . 77
4.3.2 Layer Classification and Tree Sorting . . . 78
4.3.3 Compress and Predict . . . 81
4.3.4 Structure `Sparse` . . . 85
4.3.5 Apply . . . 88

4.3.6 Gauss Quadrature and Error . . . 88
4.3.7 Choice of the Degree of Quadrature . . . 90
4.3.8 Computation of the Matrix Entries . . . 91
4.3.9 Rhs . . . 96
4.3.10 Coarse . . . 98

**5 Numerical Experiments** **99**
5.1 Parameter Study . . . 100
5.2 Laplace Problems Solved by the Single Layer Operator . . . 104
5.2.1 Fichera's Vertex . . . 105
5.2.2 Crankshaft . . . 107
5.2.3 Gearwheel . . . 109
5.3 Laplace Problems Solved by the Double Layer Operator . . . 111
5.4 Exterior Helmholtz Problems . . . 116
5.4.1 Cartoon Right-Hand Side . . . 116
5.4.2 Scattering Problems . . . 118

**6 Goal-Oriented Error Estimation** **123**
6.1 Motivation and Background . . . 123
6.2 Refinement Strategies . . . 124
6.3 Laplace Equation Solved by the Single Layer Operator . . . 126
6.4 Laplace Equation Solved by the Double Layer Operator . . . 130

# Introduction

In science and engineering one often comes across partial differential equations in three dimensions, some of which can be formulated as boundary integral equations on the boundary $\Gamma = \partial\Omega$ of the three-dimensional domain $\Omega$ of interest. Solving the original problem would result in seeking the solution in a three-dimensional domain (e.g. by means of finite element methods), which would lead to a sparse but extremely large system. Rewriting the problem as a boundary value problem not only reduces the dimensionality by one, but does give the possibility to solve also the exterior problem. Especially for exterior problems, this approach brings many advantages, since it is not necessary to find a way (e.g. the introduction of artificial boundaries) to handle the infinite expansion of the domain. However, this advantage does not come entirely without cost. First, since the involved operators are not local, the resulting matrices are dense and the complexity to assemble and solve the resulting system of linear equations is at least $\mathcal{O}(N^2)$, with $N$ denoting the degrees of freedom. Moreover, matrices which stem from the discretisation of boundary integral operators with non-zero order become more and more ill-conditioned as the degrees of freedom increase. Second, reformulating a partial differential equation in a domain into a boundary integral equation on the domain's boundary does not work unconditionally for any partial differential equation. For this conversion, one needs the knowledge of a fundamental solution, i.e. a function $k(\boldsymbol{x}, \boldsymbol{y})$ which solves the problem $\boldsymbol{A}k(\boldsymbol{x}, \boldsymbol{y}) = \delta(\boldsymbol{x} - \boldsymbol{y})$ in the distributional sense for the linear differential operator $\boldsymbol{A}$ under consideration and with $\delta$ being the Dirac functional. Thus, the method is restricted to problems for which a fundamental solution is explicitly known.

In the case when a fundamental solution is known, one can make a so-called potential ansatz, featuring a boundary integral with an unknown density $\rho$, which solves the original partial differential equation for all points $\boldsymbol{x} \in \mathbb{R}^3 \setminus \Gamma$. Notice that the boundary conditions of the original boundary value problem are not yet taken into account in the mentioned ansatz. Letting $\boldsymbol{x}$ tend towards the boundary $\Gamma$ results in an equation defined only on the boundary of the domain. This boundary integral equation can subsequently be discretised by finite elements, called boundary elements in the present context, which explains the name boundary element method. By solving this equation, we obtain the unknown density $\rho$, which then can be used to compute the solution in each point $\boldsymbol{x} \in \mathbb{R}^3 \setminus \Gamma$ by the potential ansatz. The latter process is called potential evaluation. As the computation of the solution in the domain does not occur directly, but goes through a potential evaluation after having found the density, this approach is called an indirect method.

### Boundary Element Methods

Rewriting the partial differential equation under consideration into a boundary integral equation does reduce the size of the involved system of linear equations drastically, as the domain which shall be discretised has one dimension less. However, the involved system of linear equations emerging from the discretisation of the boundary integral operator is still large and, more importantly, it is dense. Furthermore, the appearing integral operators are singular. This fact is why there has been a large amount of research in order to overcome this obstacle. Modern approaches like the *fast multipole method* [50,79], the *panel clustering* [53], the *adaptive cross approximation* [2,4], or *hierarchical matrices* [52,88] are known to reduce the complexity to log-linear or even linear cost. Some of these approaches aim at developing the involved function $k(\boldsymbol{x},\boldsymbol{y})$ into a degenerated kernel approximation, for example by polynomials. Another possibility is to use a few exact entries from the original matrix for constructing a low-rank approximation, which is the idea of the adaptive cross approximation.

### Wavelet Matrix Compression

In this thesis, we will consider a different approach, namely wavelet matrix compression, which is known to reduce the complexity, too. In the past, there has been a large amount of research concerning the use of wavelets for solving boundary integral equations. In [7], it was discovered that the kernel function $k(\boldsymbol{x},\boldsymbol{y})$ of a singular integral operator, stemming from a boundary integral equation, contains many small coefficients when using wavelets for its approximation. Entries in the system matrix which fall below a certain threshold can be neglected, i.e. they can be set to zero, yielding a so-called compressible matrix. It has been shown in [27,28,84] that all but only $\mathcal{O}(N)$ matrix entries can be neglected with the convergence rate of the underlying Galerkin scheme still being guaranteed. This procedure of neglecting non-relevant matrix entries is referred to as matrix compression.

The compression pattern of the matrix can thereby be identified in advance. To do so, we need estimates on the size of the matrix entries. The first estimate, which corresponds to wavelets having a large distance from each other, amounts to the first compression. Using the first compression alone does not yet reduce the number of entries to $\mathcal{O}(N)$. In order to being able to have a compressed system matrix with only $\mathcal{O}(N)$ entries, we need to make use of a second estimate for wavelets which overlap. Suppose that two wavelets on a largely different scale are located such that the small wavelet lies completely inside the smooth part of the large one. Then, one can exploit the distance between the singular support of the large wavelet to the support of the small one. This additional estimate amounts to the second compression. Using both of these estimates finally results in a matrix having only $\mathcal{O}(N)$ non-zero entries. Let us remark at this point that it is essential to have a wavelet basis which features sufficiently many vanishing moments.

Using wavelets for matrix compression has also been applied in many other fields such as

shape optimisation [36, 37, 55] and FEM-BEM coupling [48, 57, 58], or for solving partial differential equations with stochastic input parameters [61, 91].

### Adaptive Wavelet Schemes

Another issue to be addressed is the one of adaptivity. For some geometries or right-hand sides, it is necessary being able to resolve specific parts of the geometry, while other parts could stay coarse. One example would be a geometry featuring corners and edges. For such a domain, the solution itself admits singularities, demanding for refinement. A uniform approach leads to very large systems even if it is not necessary to choose such a fine resolution for the whole geometry. In such a situation, an adaptive refinement reduces the degrees of freedom drastically without compromising the accuracy. This means that not only a lot of computation power can be saved, but also a lot of memory, making the computation of certain problems possible in the first place.

In addition to the wavelets' beneficial compression properties, another motivation for the usage of these particular basis functions lies in the following fact. In contrast to other basis functions, the wavelets contain some information on the local regularity of a function. This means that we can use the magnitude of the coefficients in their series expansion to find out where further refinement is needed. In the finite element setting refinement is achieved by mesh refinement, where the decision of where to further refine the mesh depends on some error indicator which has been computed for the solution of the previous step. Compared to finite element methods, we cannot compute the residual exactly for boundary integral equations. Nevertheless, there exist reliable and efficient estimators for the residual [39] and also optimal convergence rates for adaptive refinement have been proven for boundary element methods [41, 45]. However, we are not aware of an implementation which combines these error estimators with fast boundary element methods.

In this thesis, we use the approach according to Cohen, Dahmen and DeVore, which can be found in [16, 17] for local operators and in [27, 47] for nonlocal operators. Instead of projecting onto a finite-dimensional subspace after the variational formulation of the operator equation has been found, it has been shown that the boundary integral equation in its variational form is equivalent to a well-posed infinite system of linear equations. As the application of this infinite matrix has to be approximated for solving the linear system of equations and computing the residual, we choose a certain portion out of this infinite matrix. Thus, in our framework, refinement rather means that more basis functions are added. To be more precise, we aim at choosing the $N$ wavelets with the largest coefficients, i.e. the wavelets which contribute most to the solution. This procedure is referred to as the best $N$-term approximation [33].

### Thesis Overview

In **Chapter 1**, we will introduce the boundary element method in general. Suppose that there is given a second order, elliptic partial differential equation in some $n$-dimensional, simply connected and bounded domain $\Omega$, supplemented by boundary conditions on the boundary $\Gamma := \partial\Omega$. Although the method itself is not restricted to a fixed spatial dimension, we will restrict ourselves in this thesis to partial differential equations in a three-dimensional domain. We can consider an interior boundary value problem, where the solution is sought inside this domain, or an exterior boundary value problem, where the solution is sought in the unbounded exterior domain. The first section of Chapter 1 will be concerned with deriving the boundary integral formulation for both, the interior and the exterior Laplace problem with Dirichlet boundary conditions as well as with Neumann boundary conditions. In the second section of Chapter 1, we introduce the exterior Helmholtz equation which arises from the wave equation after separating its solution into a spatial and a time dependent part and assuming a time-harmonic behaviour. For the exterior Helmholtz problem, we choose Dirichlet boundary conditions, which are associated with describing the propagation of a wave around a soft-sound object. In order to cast the Helmholtz problem into a boundary integral equation, we will make use of the ansatz according to Brakhage and Werner [11], as it eliminates the arising problem concerning the uniqueness of the solution for irregular wavenumbers. By choosing this ansatz we will arrive at a combination of the acoustic single layer operator and the acoustic double layer operator. For this combination, the theory presented in [27, 47] does not hold any more. However, as this operator is a compact perturbation of a symmetric and positive definite operator, the theory from [44] can be applied to prove optimality of the adaptive scheme under consideration. We will then proceed to the scattering problem, which is closely related to the Helmholtz equation, by introducing a direct and an indirect formulation for its solution. In the following section, after having all desired boundary integral formulations at hand, we discuss their variational formulation and subsequent Galerkin projection onto a finite-dimensional subspace by means of a single-scale basis. We will close Chapter 1 by giving a short overview on fast boundary element methods. Notice that, in this thesis, we will use only piecewise constant ansatz functions for discretising the associated integral equations, which is why we will not consider boundary integral equations featuring the hypersingular operator. For the treatment of boundary integral equations featuring the hypersingular operator, one needs globally continuous ansatz functions, see e.g. [63, 82, 86].

In **Chapter 2**, we are going to give an introduction to wavelets, in particular aiming to arrive at the setting which we are going to use later for our implementation. After having at hand a single-scale basis on $\mathbb{R}$, we will encounter the first wavelets on $\mathbb{R}$ by choosing appropriate linear combinations of the given single-scale functions. Afterwards, we will change our setting from $\mathbb{R}$ to the unit interval, as we wish to later arrive at wavelets on the unit square. This conversion poses certain obstructions, especially for large wavelets, as there is now a boundary preventing the wavelets to be translated unconditionally. This obstacle requires some modifications for the wavelets at the boundary. Without going too

much into detail on their construction, we introduce two wavelet bases on the unit interval which form the foundation for the subsequent construction of wavelets on the unit square. In the following section, we introduce two wavelet bases on the unit square, which are obtained by taking the tensor product of the basis functions on the unit interval. Finally, we use the constructed wavelet bases for obtaining wavelets on the surface of a domain. This is achieved by mapping the unit square onto the boundary of the three-dimensional domain, which is represented as a union of four-sided patches. We close this chapter by introducing the system of linear equations emerging after the Galerkin projection onto the finite-dimensional subspace given by a finite subset of this new basis. Notice, that there exists also the possibility for constructing wavelet bases by using higher order polynomial ansatz functions, see e.g. [18, 30, 54, 60]. However, as we develop a code for piecewise constant wavelets, we restrict the discussion to our case here.

**Chapter 3** will be concerned with the theoretical aspects of the adaptive wavelet scheme. In its first section, we motivate the use of adaptive methods when considering the numerical treatment of the previously introduced boundary integral equations. We introduce the setting for our adaptive algorithm according to the lines of the theory which has already been developed in e.g. [16, 17, 26–28, 46, 47]. The key ingredient throughout all of these articles is the equivalence of the underlying boundary integral equation in its variational form to an infinite system of linear equations. Let us remark here that this is the key difference in comparison to adaptive finite element methods. For adaptive wavelet methods, we stay in the infinite-dimensional setting, which makes the use of stability conditions, such as the *inf-sup* condition, which are necessary for the adaptive finite element setting, dispensable. However, as it is not possible to truly apply an infinite operator, we have to find a way of applying it approximately, such that optimal convergence is still guaranteed. This is achieved by restricting the underlying infinite-dimensional index set to a finite subset of a fixed cardinality $N$, which is realised by choosing the appropriate rows and columns from the underlying infinite-dimensional matrix.

Besov spaces are closely related to nonlinear approximation, which is what the second section of Chapter 3 will be concerned with. For non-smooth domains or singular right-hand sides, regularity of the solution is decreased and so may be the efficiency of the approximation [23,33]. That is why, in this context, we will need different function spaces than for the uniform setting, these are Besov spaces instead of Sobolev spaces. The condition that the solution is contained in a Besov space is much milder than demanding that it be contained in a Sobolev space. Besov spaces have been studied in the context of adaptive algorithms in e.g. [21, 23, 33]. We are interested in the question of how small the approximation error can become when choosing $N$ coefficients from the solution's expansion with respect to the wavelet basis. Also, we want the error, produced by approximation, to be the best error with respect to the chosen $N$ degrees of freedom. We thus establish the concept of the best $N$-term approximation, being a form of nonlinear approximation, which is essential in order to measure the error produced by the adaptive wavelet method. Subsequently, we introduce approximation spaces, weak $\ell^p$ spaces and, most importantly, Besov spaces.

In the third section of Chapter 3, we will focus on the building blocks of the adaptive wavelet method like the routines COARSE, RHS and APPLY. The adaptive setting makes it necessary to work with tree structures, as they are able to fully exploit the adaptivity of the method. In particular, the efficient computation of the matrix entries, which is the most demanding and time consuming part of the whole implementation, is not possible in optimal complexity without using the appropriate adaptive structures. The discussion on the theoretical details of the key routines in the implementation is preceded by an introduction on trees. The best $N$-term tree approximation is then according to the best $N$-term approximation, with the only difference that the underlying index set is restricted to a tree structure. Subsequently, we focus on each building block separately and state the purpose and the requirements for each of these blocks in order to produce an algorithm of optimal complexity.

**Chapter 4** forms the practical counterpart to Chapter 3, as it focusses on the implementation of the adaptive method. A fast implementation of the uniform wavelet method does already exist, see [54], and the realisation of routines in a uniform setting like APPLY and RHS can be found therein. While the principle realisation of assembling and solving the system of linear equations is thus nothing new, the innovation of our implementation is the conversion and fresh implementation of these routines to work with truly adaptive structures, namely trees. The efficient computation of e.g. the matrix entries is not possible without using the appropriate adaptive structures. Though the literature cited in Chapter 3 discusses the theoretical requirements for the realisation of an adaptive wavelet scheme, we are not aware of a practical realisation by truly adaptive data structures so far.

In the first two sections of Chapter 4, we will focus on the newly developed adaptive structures for representing the elements and the wavelets. In an adaptive setting, neither elements nor wavelets can efficiently be arranged into arrays any more, which is why we have to find a way to represent them uniquely by a tree. However, elements as well as wavelets have many similarities among translated and refined versions of each other. This is why we introduce a lean data structure to represent the elements and wavelets. Hence, after specifying how we represent elements and wavelets, we turn towards initialising and, more importantly, maintaining the introduced structures. After we provided these details, we will turn towards the related building blocks APPLY, RHS and COARSE of the adaptive algorithm, which are already known from the previous chapter. For these routines, we provide the implementational details, tailored to the present context of boundary element methods. Moreover, we provide some insight in the implementation of routines like COMPRESSION, PREDICTION and ASSEMBLING. The numerical method is able to compute the solution of the boundary integral equation in asymptotically optimal complexity. This means that any target accuracy can be achieved at a computational expense that stays proportional to the number of degrees of freedom (within the setting determined by the underlying wavelet basis) that would ideally be necessary for realizing that target accuracy if full knowledge about the unknown solution were given.

In **Chapter 5**, we will present several numerical results for our adaptive implementation.

In the first section of Chapter 5, we study the choice of certain parameters which appear in the implementation. After the parameters are chosen, we present numerical examples for solving the interior Laplace equation with Dirichlet boundary conditions by means of the single layer potential. Here, we will consider various geometries as well as different right-hand sides. Subsequently, in the third section of Chapter 5, we present numerical examples for solving the interior Laplace equation involving the double layer operator for its solution. We will again present examples by using different right-hand sides. Finally, we turn towards solving the exterior Helmholtz problem, where we use the ansatz according to Brakhage and Werner to cast the boundary value problem into a boundary integral equation. After presenting one numerical example for the solution of the exterior (low-frequency) Helmholtz equation, we proceed to scattering problems where we consider also various higher wavenumbers.

In the last **Chapter 6**, we present the ansatz of goal-oriented error estimation in the context of solving adaptive boundary integral equations. While in the standard adaptive approach we approximate the complete unknown density $\rho$, the idea of goal-oriented error estimation lies in approximating a linear output functional $g(\rho)$ of it. In order to approximate this quantity of interest, one might need significantly less degrees of freedom than would be necessary for approximating the complete unknown density. Instead of controlling the residual from the primal problem, one additionally introduces a dual problem and controls both residuals. In the field of goal-oriented error estimation, there has been a lot of research in the context of finite element methods, see e.g. [1,5,6,31,38,76], as well as for boundary element methods, see e.g. [40–42].

After introducing two different refinement strategies for goal-oriented error estimation, we present several numerical examples for solving the interior Laplace equation. Thereby, our first numerical examples will consider the solution of the interior Laplace equation by means of the single layer operator. Afterwards, we present also an example for the solution of the interior Laplace equation by means of the double layer operator.

We will resume this thesis with some concluding remarks as well as an outlook on a possible further continuation of this thesis.

# 1

# Boundary Element Method

In this chapter, we review the principles of the boundary element method. To this end, suppose that we are given a partial differential equation in some $n$-dimensional simply connected and bounded domain $\Omega$, supplemented by boundary conditions on the boundary $\Gamma := \partial\Omega$. We can consider an interior problem, where the boundary value problem shall be solved inside this domain. We can also consider an exterior problem, where the boundary value problem is defined in the exterior domain $\Omega^c := \mathbb{R}^n \setminus \bar{\Omega}$. As the discretisation of an $n$-dimensional domain may be expensive, especially in the case of an exterior problem, it may be desirable to convert the problem into an equivalent one on the $(n-1)$-dimensional boundary of the domain. Although the method itself is not restricted to a fixed spatial dimension, we will restrict ourselves in this thesis to partial differential equations in a three-dimensional domain.

The key idea, to convert the underlying problem into one on the boundary, is the knowledge of the so-called fundamental solution $k\colon \mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{C}$. This means that the method itself is restricted to problems of which a fundamental solution can be calculated. Let $\boldsymbol{A}$ be a linear differential operator with constant coefficients. Then, $k(\boldsymbol{x}, \boldsymbol{y})$ is its fundamental solution, if it solves the problem

$$\boldsymbol{A}k(\boldsymbol{x}, \boldsymbol{y}) = \delta(\boldsymbol{x} - \boldsymbol{y})$$

in the distributional sense, with $\delta$ denoting the Dirac functional.

With the fundamental solution at hand, one can make an ansatz by the potential

$$u(\boldsymbol{x}) = \int_\Gamma k(\boldsymbol{x}, \boldsymbol{y})\rho(\boldsymbol{y})\,\mathrm{d}\sigma_{\boldsymbol{y}} \quad \text{for } \boldsymbol{x} \in \Omega \tag{1.1}$$

with the unknown density $\rho$. It solves the underlying partial differential equation for all $\boldsymbol{x} \in \mathbb{R}^3 \setminus \Gamma$. In order to satisfy the desired boundary conditions, by letting $\boldsymbol{x}$ tend to $\Gamma$, we end up having to solve the equation

$$(\mathcal{A}\rho)(\boldsymbol{x}) = f(\boldsymbol{x}) \quad \text{for } \boldsymbol{x} \in \Gamma$$

on $\Gamma$ for some boundary integral operator $\mathcal{A}$. Discretising this boundary integral equation by finite elements, in this context called boundary elements, leads to the boundary element method.

We see that rather than directly obtaining the values in the domain, the boundary element method aims at computing a density on the boundary by using the known boundary conditions. In a next step, the integral equation (1.1) can be used to evaluate the solution at each point in the (interior or exterior) domain. There are two approaches with which one can treat the underlying problem, the direct method and the indirect method. The direct method uses the Green representation formula and the fundamental solution of the according boundary value problem to represent the solution via its potentials. The respective densities then correspond to the unknown Cauchy data. In this thesis, we lay our main focus on the indirect approach, where the density derived from the boundary integral equation is physically meaningless. We thus have to do the potential evaluation (1.1) afterwards.

This chapter is organised as follows. First, we focus on the derivation of the according boundary integral equations for the interior and exterior Laplace problem. Thereby, we consider Dirichlet boundary conditions as well as Neumann boundary conditions and introduce the appearing integral operators. Second, we consider the exterior Helmholtz equation for Dirichlet boundary conditions, describing the time-harmonic propagation of a wave around a sound-soft object. We derive the according boundary integral equation by using the ansatz of Brakhage and Werner, after having addressed the difficulties arising for irregular wavenumbers. Subsequently, we focus on the scattering problem, which is closely connected to the Helmholtz equation. For the scattering problem with a sound-soft scatterer, we introduce the direct and indirect formulation of the problem by a boundary integral equation. The section will be completed by reviewing the variational formulation and the Galerkin projection onto a finite dimensional subspace for the encountered boundary integral equations.

## 1.1 Laplace Equation

In this section, we consider the reformulation of the Laplace equation in three dimensions by boundary integral equations. In particular, we will derive the boundary integral equations arising from the boundary value problem with both, Dirichlet as well as Neumann boundary conditions. Since, in this thesis, we will use only piecewise constant ansatz functions for the upcoming discretisation of the associated integral equations, we will not consider boundary integral equations where the hypersingular operator appears. For the treatment of this integral operator, one needs globally continuous ansatz functions, see e.g. [63,82,86].

### 1.1.1 Dirichlet Problem for the Laplace Equation

Consider the Laplace equation with Dirichlet boundary conditions in some bounded and simply connected domain

$$\begin{aligned} \Delta u &= 0 \quad \text{in } \Omega, \\ u &= f \quad \text{on } \Gamma, \end{aligned} \tag{1.2}$$

i.e., we seek a harmonic function $u \in H^1(\Gamma)$ with given Dirichlet data $f \in H^{1/2}(\Gamma)$. To solve the interior Dirichlet problem (1.2), we introduce the kernel

$$k(\boldsymbol{x}, \boldsymbol{y}) \coloneqq \frac{1}{4\pi \|\boldsymbol{x} - \boldsymbol{y}\|},$$

which is the fundamental solution of the Laplacian. The single layer operator

$$(\mathcal{S}\rho)(\boldsymbol{x}) \coloneqq \frac{1}{4\pi} \int_\Gamma \frac{1}{\|\boldsymbol{x} - \boldsymbol{y}\|} \rho(\boldsymbol{y}) \, \mathrm{d}\sigma_{\boldsymbol{y}} = f(\boldsymbol{x}), \quad \boldsymbol{x} \in \Gamma, \tag{1.3}$$

is symmetric, positive definite and of order $-1$, that is $\mathcal{S}\colon H^{-1/2}(\Gamma) \to H^{1/2}(\Gamma)$, see e.g. [51, 63, 71]. With this operator at hand, we can formulate the boundary integral equation

$$\mathcal{S}\rho = f \quad \text{on } \Gamma \tag{1.4}$$

for the unknown density $\rho \in H^{-1/2}(\Gamma)$. As soon as the density is calculated, the solution $u$ of the Dirichlet problem (1.2) can be found by evaluating

$$u(\boldsymbol{x}) = \int_\Gamma k(\boldsymbol{x}, \boldsymbol{y}) \, \rho(\boldsymbol{y}) \, \mathrm{d}\sigma_{\boldsymbol{y}} \quad \text{for } \boldsymbol{x} \in \Omega. \tag{1.5}$$

In particular, the density $\rho$ and thus the solution $u$ to the interior Dirichlet problem is unique, see [51,71].

Instead of the interior Laplace equation (1.2), we can also consider the exterior problem

$$\begin{aligned} \Delta u &= 0 \quad \text{in } \Omega^c, \\ u &= f \quad \text{on } \Gamma, \end{aligned} \tag{1.6}$$

with $\Omega^c := \mathbb{R}^3 \setminus \bar{\Omega}$ being the exterior of $\Omega$. We seek then the solution $u \in H^1_{\text{loc}}(\Omega^c)$, where $H^1_{\text{loc}}(\Omega^c)$ denotes the space of all functions $v \in L^2(\Omega^c)$ which are locally contained in $H^1$. For the exterior problem, we additionally require the decay

$$|u(\boldsymbol{x})| = \mathcal{O}\left(\frac{1}{\|\boldsymbol{x}\|}\right) \quad \text{as } \|\boldsymbol{x}\| \to \infty.$$

Likewise to the interior problem, solving (1.4) for the density $\rho \in H^{1/2}(\Gamma)$ gives the solution to (1.6) via the potential evaluation (1.5) for all $\boldsymbol{x} \in \Omega^c$.

Another possibility for solving the Dirichlet problem for the Laplace equation is to introduce the double layer operator

$$(\mathcal{D}\rho)(\boldsymbol{x}) \;:=\; \frac{1}{4\pi}\int_\Gamma \frac{\langle \boldsymbol{x}-\boldsymbol{y}, \boldsymbol{n}_{\boldsymbol{y}}\rangle}{\|\boldsymbol{x}-\boldsymbol{y}\|^3}\,\rho(\boldsymbol{y})\,\mathrm{d}\sigma_{\boldsymbol{y}} = \int_\Gamma \frac{\partial}{\partial \boldsymbol{n}_{\boldsymbol{y}}} k(\boldsymbol{x},\boldsymbol{y})\,\rho(\boldsymbol{y})\,\mathrm{d}\sigma_{\boldsymbol{y}}, \quad \boldsymbol{x} \in \Gamma. \tag{1.7}$$

Here, $\boldsymbol{n}_{\boldsymbol{y}}$ denotes the normal in the point $\boldsymbol{y} \in \Gamma$, oriented to the outside of $\Omega$, and $k(\boldsymbol{x},\boldsymbol{y})$ denotes the fundamental solution as before. Due to the jump condition of the associated double layer potential, when $\boldsymbol{x}$ approaches the boundary $\Gamma$, cp. [82, 86], we end up this time with the integral equation

$$\left(\frac{1}{2}\mathbf{I} \pm \mathcal{D}\right)\rho = f \quad \text{on } \Gamma,$$

where $\frac{1}{2}\mathbf{I} \pm \mathcal{D}\colon L^2(\Gamma) \to L^2(\Gamma)$ is an operator of order 0. Dependent on whether we want to solve the exterior or the interior problem, we choose the appropriate leading sign in the above equation; " $+$ " for the exterior problem and " $-$ " for the interior problem. The density of the interior problem is unique [51, 71], while we need to impose the following two conditions in order to ensure uniqueness for the exterior problem:

$$\int_\Gamma f(\boldsymbol{x})\mathrm{d}\sigma_{\boldsymbol{x}} = 0 \quad \text{and} \quad \int_\Gamma \rho(\boldsymbol{x})\mathrm{d}\sigma_{\boldsymbol{x}} = 0.$$

This issues from the fact that the value $-1/2$ is an eigenvalue of $\mathcal{D}$ with the constant functions as eigenvectors and hence $\frac{1}{2}\mathbf{I}+\mathcal{D}$ is singular. Having finally the density at hand, the solution of the Laplace equation can be calculated by the formula

$$u(\boldsymbol{x}) = \int_\Gamma \frac{\partial}{\partial \boldsymbol{n}_{\boldsymbol{y}}} k(\boldsymbol{x},\boldsymbol{y})\,\rho(\boldsymbol{y})\,\mathrm{d}\sigma_{\boldsymbol{y}} \tag{1.8}$$

for all $\boldsymbol{x} \in \Omega$ and $\boldsymbol{x} \in \Omega^c$, respectively.

### 1.1.2 Neumann Problem for the Laplace Equation

The next problem we consider is the Laplace equation with Neumann boundary conditions

$$\begin{aligned} \Delta u &= 0 \qquad \text{in } \Omega \text{ or in } \Omega^c, \\ \frac{\partial u}{\partial \boldsymbol{n}} &= g \qquad \text{on } \Gamma, \end{aligned} \tag{1.9}$$

where we seek the unknown solution $u \in H^1(\Omega)$ or $u \in H^1_{\mathrm{loc}}(\Omega^c)$ for a given right-hand side $g \in H^{-1/2}(\Gamma)$. Similar as for the Dirichlet problem, we can choose $\Omega$ or $\Omega^c$, resulting in the interior or the exterior Neumann problem, respectively. In the exterior case, again the decay property

$$|u(\boldsymbol{x})| = \mathcal{O}\left(\frac{1}{\|\boldsymbol{x}\|}\right) \quad \text{for } \|\boldsymbol{x}\| \to \infty$$

is required.

We can solve the Neumann problem by means of the adjoint double layer operator

$$(\mathcal{D}^t \rho)(\boldsymbol{x}) \;:=\; \frac{1}{4\pi} \int_\Gamma \frac{\langle \boldsymbol{x}-\boldsymbol{y}, \boldsymbol{n}_{\boldsymbol{x}} \rangle}{\|\boldsymbol{x}-\boldsymbol{y}\|^3} \, \rho(\boldsymbol{y}) \, \mathrm{d}\sigma_{\boldsymbol{y}} = \int_\Gamma \frac{\partial}{\partial \boldsymbol{n}_{\boldsymbol{x}}} k(\boldsymbol{x}, \boldsymbol{y}) \, \rho(\boldsymbol{y}) \, \mathrm{d}\sigma_{\boldsymbol{y}}, \quad \boldsymbol{x} \in \Gamma, \tag{1.10}$$

which amounts to the boundary integral equation

$$\left(\mathcal{D}^t \pm \frac{1}{2}\mathbf{I}\right) \rho = g \quad \text{on } \Gamma.$$

As in the case of the double layer operator, the choice of the leading sign depends again on whether we solve the exterior or the interior problem. This integral operator is also of order zero, that is $\mathcal{D}^t \pm \frac{1}{2}\mathbf{I} \colon L^2(\Gamma) \to L^2(\Gamma)$. Note that, when solving the interior problem, we have to demand that

$$\int_\Gamma g(\boldsymbol{x}) \, \mathrm{d}\sigma_{\boldsymbol{x}} = 0 \quad \text{and} \quad \int_\Gamma \rho(\boldsymbol{x}) \, \mathrm{d}\sigma_{\boldsymbol{x}} = 0$$

in order to ensure unique solvability [51, 71].

Finally, the solution $u$ is calculated from the unknown density by evaluating

$$u(\boldsymbol{x}) = \int_\Gamma k(\boldsymbol{x}, \boldsymbol{y}) \, \rho(\boldsymbol{y}) \, \mathrm{d}\sigma_{\boldsymbol{y}}$$

for all $\boldsymbol{x} \in \Omega$ and $\boldsymbol{x} \in \Omega^c$.

## 1.2 Helmholtz Equation

The next problems we want to consider are related with the Helmholtz equation. We will first introduce the Helmholtz equation by describing how it arises from the wave equation

and for what situations. We will encounter the Sommerfeld radiation condition and see its importance for ensuring the uniqueness of the solution. After the Helmholtz equation is properly introduced, we will convert it into a boundary integral equation. We will especially encounter the problem of irregular wavenumbers and introduce the ansatz of Brakhage and Werner [11] in order to overcome this problem. Next, we will discuss the scattering of acoustic waves by an obstacle which is surrounded by a homogeneous medium. We will derive the direct as well as the indirect formulation for solving this scattering problem. Both approaches we are about to introduce are well known and frequently used [19, 20, 49, 56]. In scattering theory, there can be distinguished between objects that are penetrable and such that are impenetrable. For this thesis, we restrict ourselves to the discussion on impenetrable objects. Also we do only consider the forward scattering problem. For further details on these subjects and also the treatment of the inverse scattering problem, the reader is referred to [19, 20].

### 1.2.1 Sommerfeld's Radiation Condition

We start this section by explaining how the Helmholtz equation arises from the wave equation in the following form:

$$\frac{\partial U(\boldsymbol{x},t)^2}{\partial t^2} - c^2 \Delta U(\boldsymbol{x},t) = 0. \tag{1.11}$$

Here, $c$ denotes the speed of the wave, which is assumed to be space independent. Readers, which are interested in the physical background on how equation (1.11) is obtained, can find a detailed derivation in e.g. [19, 49]. Next, we use the ansatz of separation of variables

$$U(\boldsymbol{x},t) = X(\boldsymbol{x}) \cdot T(t),$$

where we assume that the function can be decomposed into a solely space dependent part $X(\boldsymbol{x})$ and a solely time dependent part $T(t)$. Furthermore, we assume that the function $U(\boldsymbol{x},t)$ is time harmonic, meaning that

$$U(\boldsymbol{x},t) = \Re\left(u(\boldsymbol{x})\, e^{-i\omega t}\right)$$

with $\omega > 0$ being the frequency. Inserting this ansatz into the original equation (1.11) gives us

$$u(\boldsymbol{x})\,(i^2\omega^2)\, e^{-i\omega t} - c^2 \Delta u(\boldsymbol{x})\, e^{-i\omega t} = 0.$$

After factoring out the term $e^{-i\omega t}$ and reordering, we notice that the stationary part $u(\boldsymbol{x})$ fulfils the Helmholtz equation

$$\Delta u(\boldsymbol{x}) + \kappa^2 u(\boldsymbol{x}) = 0 \tag{1.12}$$

in the exterior domain $\Omega^c$, where we set $\kappa = \omega/c = 2\pi/\lambda > 0$. The parameter $\kappa$ is called the wavenumber and the parameter $\lambda$ is called the wavelength, physically describing the oscillation of the wave.

**Definition 1.** A solution $u$ to the Helmholtz equation is called radiating if

$$\lim_{r\to\infty} r\left(\frac{\partial u}{\partial r} - i\kappa u\right) = 0 \quad \text{where } r = \|\boldsymbol{x}\|$$

holds uniformly in all directions $\boldsymbol{x}/\|\boldsymbol{x}\|$.

This radiation condition, introduced by Sommerfeld in [85], is known as the Sommerfeld radiation condition. It is necessary in order to ensure the uniqueness of the solution, which can be seen as follows. We consider the Laplace operator in spherical coordinates

$$\Delta u(r,\vartheta,\varphi) = \frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\frac{\partial u}{\partial r}\right) + \frac{1}{\sin(\vartheta)r^2}\frac{\partial}{\partial \vartheta}\left(\sin(\vartheta)\frac{\partial u}{\partial \vartheta}\right) + \frac{1}{r^2\sin^2(\vartheta)}\frac{\partial^2 f}{\partial \varphi^2}.$$

For a function $u = u(r)$, the last two terms become zero and we have

$$\Delta u + \kappa^2 u = \frac{\partial^2 u}{\partial r^2} + \frac{2}{r}\frac{\partial u}{\partial r} + \kappa^2 u = \frac{\partial^2}{\partial r^2}(ru) + \kappa^2(ru) = 0.$$

Considering the last equality, we have the solutions

$$u_1(r) = \frac{e^{i\kappa r}}{r} \quad \text{and} \quad u_2(r) = \frac{e^{-i\kappa r}}{r}.$$

We observe that

$$\begin{aligned}
\lim_{r\to\infty} r\left(\frac{\partial u_1}{\partial r} - i\kappa u_1\right) &= \lim_{r\to\infty} r\left(i\kappa u_1 - \frac{1}{r}u_1 - i\kappa u_1\right) \quad = 0,\\
\lim_{r\to\infty} r\left(\frac{\partial u_2}{\partial r} - i\kappa u_2\right) &= \lim_{r\to\infty} r\left(-i\kappa u_2 - \frac{1}{r}u_2 - i\kappa u_2\right) \neq 0.
\end{aligned}$$

Thus, only $u_1(r) = \frac{e^{i\kappa r}}{r}$ fulfils the Sommerfeld radiation condition. The physical interpretation is that $u_1$ is an outgoing wave, see e.g. [19], since the scattered wave is generated at time $t = 0$ by the object $\Omega$.

**Remark 2.** In Sommerfeld's original work [85], it is demanded that, aside from the radiation condition from Definition 1, there shall hold the so-called finiteness condition

$$u(\boldsymbol{x}) = \mathcal{O}\left(\frac{1}{\|\boldsymbol{x}\|}\right) \tag{1.13}$$

uniformly in all directions $\boldsymbol{x}/\|\boldsymbol{x}\|$ together with the existence of Green's function in the exterior domain. In a later work [73], Magnus proves uniqueness without the existence of Green's function. One year later, Rellich proved in [78] a stronger result on uniqueness, which additionally omits the use of (1.13). Finally, in 1956, Wilcox proves uniqueness in [92], as well as the derivation of representation formula (1.15) for a complex wavenumber $\kappa$. Thereby, he used a weaker condition instead of the original radiation condition from Definition 1.

### 1.2.2 Reformulation as a Boundary Integral Equation

Consider the exterior Helmholtz problem

$$\begin{aligned} \Delta u + \kappa^2 u &= 0 \quad \text{in } \Omega^c, \\ u &= f \quad \text{on } \Gamma, \\ \lim_{r\to\infty} r\left(\frac{\partial u}{\partial r} - i\kappa u\right) &= 0 \quad \text{as } r = \|\boldsymbol{x}\| \to \infty. \end{aligned} \tag{1.14}$$

As boundary conditions, we chose Dirichlet boundary conditions $u = f$ at $\Gamma$, in which case the right-hand side $f$ is in $H^{1/2}(\Gamma)$. Other possibilities are Neumann boundary conditions, that is $\partial u/\partial \boldsymbol{n} = f$, or impedance boundary conditions, that is $i\kappa\lambda u + \partial u/\partial \boldsymbol{n} = f$. We go into more detail on the physical meaning of the different boundary conditions in Section 1.2.3, where we introduce the scattering problem.

Before we turn to reformulating problem (1.14) into a boundary integral equation, we want to state the representation theorem for a solution to the Helmholtz equation. To this end, by means of Green's second theorem, the following result can be shown:

**Theorem 3.** For a solution $u$ to the exterior Helmholtz equation, which satisfies the Sommerfeld radiation condition, there holds:

$$u(\boldsymbol{x}) = \int_\Gamma \left( u(\boldsymbol{y}) \frac{\partial k(\boldsymbol{x},\boldsymbol{y})}{\partial \boldsymbol{n}_{\boldsymbol{y}}} - \frac{\partial u}{\partial \boldsymbol{n}}(\boldsymbol{y}) k(\boldsymbol{x},\boldsymbol{y}) \right) \mathrm{d}\sigma_{\boldsymbol{y}}, \quad \boldsymbol{x} \in \Omega^c. \tag{1.15}$$

Thus, $u$ can be represented by a combination of the acoustic single layer potential

$$(\mathcal{S}\rho)(\boldsymbol{x}) = \int_\Gamma k(\boldsymbol{x},\boldsymbol{y})\, \rho(\boldsymbol{y})\, \mathrm{d}\sigma_{\boldsymbol{y}} \quad \text{for all } \boldsymbol{x} \in \mathbb{R}^3 \setminus \Gamma \tag{1.16}$$

and the acoustic double layer potential

$$(\mathcal{D}\rho)(\boldsymbol{x}) = \int_\Gamma \frac{\partial k(\boldsymbol{x},\boldsymbol{y})}{\partial \boldsymbol{n}_{\boldsymbol{y}}}\, \rho(\boldsymbol{y})\, \mathrm{d}\sigma_{\boldsymbol{y}} \quad \text{for all } \boldsymbol{x} \in \mathbb{R}^3 \setminus \Gamma. \tag{1.17}$$

The expression $k(\boldsymbol{x},\boldsymbol{y})$ denotes the fundamental solution of the Helmholtz equation and is of the form cf. [19]

$$k(\boldsymbol{x},\boldsymbol{y}) = \frac{1}{4\pi} \frac{e^{i\kappa\|\boldsymbol{x}-\boldsymbol{y}\|}}{\|\boldsymbol{x}-\boldsymbol{y}\|} \quad \text{for all } \boldsymbol{x} \neq \boldsymbol{y}. \tag{1.18}$$

The fact that this fundamental solution fulfils the Helmholtz equation for any $\boldsymbol{x} \in \mathbb{R} \setminus \boldsymbol{y}$, given a fixed $\boldsymbol{y}$, can be verified by differentiation.

Now, we would like to introduce two boundary integral equations which arise from the single layer potential and the double layer potential, respectively, in the exterior domain. The two equations are

$$\mathcal{S}\rho = f \quad \text{on } \Gamma \tag{1.19}$$

and

$$\left(\frac{1}{2}\mathbf{I} + \mathcal{D}\right)\rho = f \quad \text{on } \Gamma. \tag{1.20}$$

For particular values of $\kappa$, we will run into problems, since the homogeneous interior Dirichlet and Neumann boundary value problems feature nontrivial solutions [19,68,83]. According to the second theorem of Fredholm we state:

**Theorem 4.** Consider two compact, adjoint operators $\mathcal{A}, \mathcal{A}'$.

- The nonhomogeneous equation $(\mathbf{I} - \mathcal{A})\rho = f$ is solvable, if and only if the condition $\langle f, \rho' \rangle = 0$ holds for the adjoint homogeneous equation $(\mathbf{I} - \mathcal{A}')\rho' = 0$.
- The nonhomogeneous equation $(\mathbf{I} - \mathcal{A}')\rho' = f'$ is solvable, if and only if the condition $\langle \rho, f' \rangle = 0$ holds for the adjoint homogeneous equation $(\mathbf{I} - \mathcal{A})\rho = 0$.

Notice that the exterior Dirichlet problem $(\frac{1}{2}\boldsymbol{I} + \mathcal{D})\rho = f$ and the interior Neumann problem $(\frac{1}{2}\boldsymbol{I} + \mathcal{D}^t)\rho = g$ are adjoint to each other. In accordance with [19] we have the following theorem:

**Theorem 5.** The exterior Dirichlet problem (1.14) is uniquely solvable if $\kappa^2$ is not an interior Neumann eigenvalue. If $\kappa^2$ is an eigenvalue, then the integral equation is solvable, but the solution is not unique.

To overcome this problem, Brakhage and Werner [11], Leis [72] and Panich [77] came up with the idea to combine the single layer potential and the double layer potential according to

$$u = (\mathcal{D} - i\eta\mathcal{S})\,\rho$$

for some $\eta > 0$. Taking the trace leads us to the integral equation

$$\mathcal{L}\rho := \left(\frac{1}{2}\mathbf{I} + \mathcal{D} - i\eta\mathcal{S}\right)\rho = f \quad \text{on } \Gamma. \tag{1.21}$$

The underlying integral operator is of order zero, i.e. $\frac{1}{2}\mathbf{I} + \mathcal{D} - i\eta\mathcal{S}\colon L^2(\Gamma) \to L^2(\Gamma)$. It is invertible for all $\eta > 0$ and $\kappa > 0$ [11], provided that the boundary $\Gamma$ is Lipschitz [13,75]. For practical use, the coupling parameter $\eta$ is usually chosen as $\eta \sim \kappa$. In our implementation, we set it to $\eta = \kappa/2$. Further discussions on the proper choice of $\eta$ can be found in [14,70].

After the density $\rho$ in equation (1.21) is found, we can calculate the solution $u$ to the Helmholtz equation by evaluating

$$u(\boldsymbol{x}) = \int_\Gamma \left(\frac{\partial k(\boldsymbol{x},\boldsymbol{y})}{\partial \boldsymbol{n_y}} - i\,\eta\,k(\boldsymbol{x},\boldsymbol{y})\right)\rho(\boldsymbol{y})\,\mathrm{d}\sigma_{\boldsymbol{y}}, \quad \boldsymbol{x} \in \Omega^c.$$

### 1.2.3 Scattering Problems

The Helmholtz equation can be used to describe acoustic scattering at a three-dimensional object $\Omega \subset \mathbb{R}^3$. Then, we are interested in solving the Helmholtz equation in the exterior domain $\Omega^c$. We consider the situation, where we have $u = u^i + u^s$, with $u^i$ being a known incident plane wave $u^i(\boldsymbol{x}) = \exp(i\kappa\langle \boldsymbol{x}, \boldsymbol{d}\rangle)$ and $u^s$ being the scattered wave. The normalised vector $\boldsymbol{d}$ describes the direction of the plane wave. Given the direction $\boldsymbol{d}$ and the obstacle $\Omega$ with boundary $\Gamma$, we obtain the following problem for the total field $u(\boldsymbol{x}) = \exp(i\kappa\langle \boldsymbol{x}, \boldsymbol{d}\rangle) + u^s(\boldsymbol{x})$:

$$\begin{aligned} \Delta u + \kappa^2 u &= 0 \quad \text{in } \Omega^c, \\ u &= 0 \quad \text{on } \Gamma, \\ \lim_{r\to\infty} r\left(\frac{\partial u^s}{\partial r} - i\kappa u^s\right) &= 0 \quad \text{for } r = \|\boldsymbol{x}\| \to \infty. \end{aligned} \tag{1.22}$$

Notice that the boundary condition on $\Gamma$ in (1.22) is chosen as homogeneous Dirichlet boundary condition. Depending on the characteristics of the scatterer, the boundary conditions have to be set appropriately. One example is a sound-soft scatterer, which means that the pressure on the boundary is assumed to be zero. Then, the solution $u$ satisfies the Helmholtz equation with homogeneous Dirichlet conditions and the scattered wave $u^s$ is a solution to the Helmholtz equation with the boundary conditions $f = -u^i$. The other example is a sound-hard scatterer, meaning that the Neumann derivative on the boundary is zero. Then, $u$ features homogeneous Neumann boundary conditions and $u^s$ is the solution to the Helmholtz equation with the boundary conditions $f = -\partial u^i/\partial \boldsymbol{n}$. In this thesis, we will consider Dirichlet boundary conditions, i.e. a sound-soft scatterer. In order to solve the scattering problem, we will now present two possibilities on how we can proceed.

The first way is again to combine the single and double layer operator according to the ansatz of Brakhage and Werner to represent the scattered field $u^s$:

$$u^s(\boldsymbol{x}) = \int_\Gamma \frac{\partial k(\boldsymbol{x}, \boldsymbol{y})}{\partial \boldsymbol{n_y}} \rho(\boldsymbol{y}) \mathrm{d}\sigma_{\boldsymbol{y}} - i\eta \int_\Gamma k(\boldsymbol{x}, \boldsymbol{y}) \rho(\boldsymbol{y}) \mathrm{d}\sigma_{\boldsymbol{y}}, \quad \boldsymbol{x} \in \Omega^c. \tag{1.23}$$

It is shown in e.g. [14] that the function $u^s$, given by this ansatz, satisfies the scattering problem if and only if $\rho$ satisfies the boundary integral equation (1.21). Thereby, the right-hand side is given by $f := -u^i$ as mentioned above. Having the solution $\rho$ at hand, the scattered wave $u^s$ can be found by evaluating the integrals in formula (1.23).

The second possibility to obtain an integral equation involves the use of the representation formula (1.15) and Green's second theorem. With (1.15) we obtain

$$u^s(\boldsymbol{x}) = \int_\Gamma \left( u^s \frac{\partial k(\boldsymbol{x}, \boldsymbol{y})}{\partial \boldsymbol{n_y}} - \frac{\partial u^s}{\partial \boldsymbol{n}}(\boldsymbol{y}) k(\boldsymbol{x}, \boldsymbol{y}) \right) \mathrm{d}\sigma_{\boldsymbol{y}}, \quad \boldsymbol{x} \in \Omega^c, \tag{1.24}$$

for the scattered field $u^s$. Green's second theorem used for $k(\boldsymbol{x},\boldsymbol{y})$ and $u^i$ leads to

$$\int_\Gamma \left( u^i \frac{\partial k(\boldsymbol{x},\boldsymbol{y})}{\partial \boldsymbol{n}_{\boldsymbol{y}}} - \frac{\partial u^i}{\partial \boldsymbol{n}}(\boldsymbol{y}) k(\boldsymbol{x},\boldsymbol{y}) \right) \mathrm{d}\sigma_{\boldsymbol{y}} = 0, \quad \boldsymbol{x} \in \Omega^c. \tag{1.25}$$

Adding equation (1.25) to equation (1.24) and reordering gives us

$$u^s(\boldsymbol{x}) = \int_\Gamma \left( \left(u^s + u^i\right) \frac{\partial k(\boldsymbol{x},\boldsymbol{y})}{\partial \boldsymbol{n}_{\boldsymbol{y}}} - \left( \frac{\partial u^s}{\partial \boldsymbol{n}}(\boldsymbol{y}) + \frac{\partial u^i}{\partial \boldsymbol{n}}(\boldsymbol{y}) \right) k(\boldsymbol{x},\boldsymbol{y}) \right) \mathrm{d}\sigma_{\boldsymbol{y}}, \quad \boldsymbol{x} \in \Omega^c.$$

The first part under the integral vanishes due to the boundary condition $u = u^i + u^s = 0$ for sound-soft scatterers, leaving only the second part. Using the relation $u = u^i + u^s$ inside the domain $\Omega^c$, implies that the total field $u$ can be represented as

$$u(\boldsymbol{x}) = u^i(\boldsymbol{x}) - \int_\Gamma k(\boldsymbol{x},\boldsymbol{y}) \frac{\partial u}{\partial \boldsymbol{n}}(\boldsymbol{y}) \mathrm{d}\sigma_{\boldsymbol{y}}, \quad \boldsymbol{x} \in \Omega^c. \tag{1.26}$$

This equation features a new unknown, namely the Neumann data $\partial u/\partial \boldsymbol{n}$. By taking the normal derivative of (1.26) and applying again the ansatz of Brakhage and Werner, we get the boundary integral equation for obtaining the Neumann data of the total field:

$$\mathcal{L}^t \frac{\partial u}{\partial \boldsymbol{n}} = \left( \frac{1}{2}\mathbf{I} + \mathcal{D}^t - i\eta \mathcal{S} \right) \frac{\partial u}{\partial \boldsymbol{n}} = \frac{\partial u^i}{\partial \boldsymbol{n}} - i\eta u^i \quad \text{on } \Gamma.$$

This equation is again uniquely solvable for every $\eta > 0$ and $\kappa > 0$. We notice that the operator $\mathcal{L}^t$ is the transpose of $\mathcal{L}$, since $\mathcal{D}^t$ is the transpose of $\mathcal{D}$ and $\mathcal{S}$ is symmetric. Both operators $\mathcal{L}$ and $\mathcal{L}^t$ are bijections for $\eta > 0$ and $\kappa > 0$, which implies that their inverses are bounded as well.

## 1.3 Variational Formulation and Galerkin Scheme

We want to conclude this chapter by reviewing the concept of converting a boundary integral equation into its variational form. Therefore, we first consider a boundary integral equation in general form

$$\mathcal{A}\rho = f \quad \text{in } \Gamma \tag{1.27}$$

with the right-hand side $f \in H^{-s}(\Gamma)$, the unknown density $\rho \in H^s(\Gamma)$, and the boundary integral operator $\mathcal{A}\colon H^s(\Gamma) \to H^{-s}(\Gamma)$ of order $2s$. Here, $H^s(\Gamma)$ denotes the Sobolev space of possibly complex functions $f\colon \Gamma \to \mathbb{C}$ of smoothness $s$. In some cases, we additionally have the constraints

$$\int_\Gamma f(\boldsymbol{x}) \mathrm{d}\sigma_{\boldsymbol{x}} = 0 \quad \text{and} \quad \int_\Gamma \rho(\boldsymbol{x}) \mathrm{d}\sigma_{\boldsymbol{x}} = 0.$$

The variational formulation of (1.27) then reads: Find $\rho \in H^s(\Gamma)$ such that

$$\langle \mathcal{A}\rho, v\rangle = \langle f, v\rangle \text{ for all } v \in H^s(\Gamma), \tag{1.28}$$

with the additional equations $\langle f, 1\rangle = 0$ and $\langle \rho, 1\rangle = 0$ if required.

To arrive at a finite-dimensional setting, we use the Galerkin scheme, meaning that we replace the energy space $H^s(\Gamma)$ by a finite-dimensional trial space $V_j \subset H^s(\Gamma)$. This leads to the discrete formulation: Find $\rho_j \in V_j$ such that

$$\langle \mathcal{A}\rho_j, v_j\rangle = \langle f, v_j\rangle \text{ for all } v_j \in V_j. \tag{1.29}$$

We consider the trial spaces $V_j$ to be piecewise continuous polynomials of order $r$, nested $V_0 \subset V_1 \subset \cdots \subset H^s(\Gamma)$ and dense in $H^s(\Gamma)$. Here, the integer $j$ in the subscript encodes the fineness of the discretisation.

We shall assume that $\mathcal{A}$ has the form $\mathcal{M}+\mathcal{N}$, with $\mathcal{M}$ elliptic and $\mathcal{N}$ compact. Furthermore, $\mathcal{A}$ is injective and satisfies a Gårding inequality, i.e. there exists a constant $c > 0$ and a compact operator $\mathcal{B}\colon H^s(\Gamma) \to H^{-s}(\Gamma)$ such that

$$|\langle \mathcal{A}\rho, \rho\rangle + \langle \mathcal{B}\rho, \rho\rangle| \geq c\|\rho\|^2 \quad \text{for all } \rho \in H^s(\Gamma). \tag{1.30}$$

If we want to consider a problem featuring the double layer operator or Helmholtz problems we have a situation exactly as described. The operator $\mathcal{A}$ features a part $\mathbf{I}$ which is elliptic and a second part consisting of the double layer operator or a combination of the single layer operator and the double layer operator, which is compact on smooth surfaces, i.e. $\mathcal{C}^{1,\alpha}$ with $\alpha > 1$. Following the literature of Sauter and Schwab [83], these conditions imply the requirements of the following theorem, which guarantees uniqueness of the solution $\rho$, the discrete solution $\rho_j$, as well as an error estimate for the following two variational problems.

For the operator $\mathcal{A} = \mathcal{M} + \mathcal{N}$, the variational form of (1.28) reads: Find $\rho \in H^s$ such that

$$\langle \mathcal{M}\rho, v\rangle + \langle \mathcal{N}\rho, v\rangle = \langle f, v\rangle \quad \text{for all } v \in H^s. \tag{1.31}$$

With its discrete counterpart: Find $\rho_j \in V_j$ such that

$$\langle \mathcal{M}\rho_j, v_j\rangle + \langle \mathcal{N}\rho_j, v_j\rangle = f(v_j) \quad \text{for all } v_j \in V_j. \tag{1.32}$$

**Theorem 6.** ([83, Theorem 4.2.9]) Let there be given a Hilbert space and $(V_j)_j$ a dense sequence of finite-dimensional subspaces. Suppose that there holds:

(i) $\mathcal{M}$ is elliptic, i.e. there exists a constant $\alpha > 0$ such that

$$|\langle \mathcal{M}\rho, \rho\rangle| \geq \alpha\|\rho\|_{H^s}^2 \quad \text{for all } \rho \in H^s.$$

(ii) The operator $\mathcal{N}$ is compact.

(iii) For $f = 0$, the problem has only the trivial solution, i.e.

$$\langle \mathcal{M}\rho, v\rangle + \langle \mathcal{N}\rho, v\rangle = 0 \Rightarrow \rho = 0.$$

Then, the problem (1.31) has a unique solution $\rho \in H^s$ for every right-hand side $f \in H^{-s}$.

There exists a constant $j_0 > 0$ such that for all $j \geq j_0$ the Galerkin equation (1.32) has a unique solution $\rho_j \in V_j$. The sequence $(\rho_j)_j$ of solutions converges towards $\rho$ and for $j \geq j_0$ there holds the error estimate

$$\|\rho - \rho_j\|_{H^s} \leq C \min_{v_j \in V_j} \|\rho - v_j\|_{H^s}, \tag{1.33}$$

with the constant $C$ being independent of $j$.

The proof to this theorem can be found in [83].

In the case when we consider the Helmholtz equation, the constant $C$ in the error estimate (1.33) additionally depends on the wavenumber $\kappa$ and the coupling parameter $\eta$. More details on this dependence can be found for example in [14, 49].

The next step is to choose a suitable basis $\boldsymbol{\Phi}_j = (\phi_i)_i$ of $V_j$. Making the ansatz

$$\rho_j = \boldsymbol{\Phi}_j \boldsymbol{\rho}^{\boldsymbol{\Phi}_j} = \sum_i \rho_i^{\boldsymbol{\Phi}_j} \phi_{i,j}, \tag{1.34}$$

as it is well known from the finite element setting, leads to the system of linear equations

$$\boldsymbol{A}^{\boldsymbol{\Phi}_j} \boldsymbol{\rho}_j^{\boldsymbol{\Phi}_j} = \boldsymbol{f}^{\boldsymbol{\Phi}_j} \tag{1.35}$$

with the right-hand side $\boldsymbol{f}^{\boldsymbol{\Phi}_j}$ and the stiffness matrix $\boldsymbol{A}^{\boldsymbol{\Phi}_j}$ defined through

$$\boldsymbol{f}_i^{\boldsymbol{\Phi}_j} \coloneqq \langle f, \phi_i\rangle \quad \text{and} \quad \boldsymbol{A}_{i,i'}^{\boldsymbol{\Phi}_j} \coloneqq \langle \mathcal{A}\phi_{i'}, \phi_i\rangle. \tag{1.36}$$

First, the form of the matrix $\boldsymbol{A}^{\boldsymbol{\Phi}_j}$ heavily depends on properties of the underlying operator $\mathcal{A}$, more precisely on the involved kernel function. For example, in the case of the single layer operator, we end up with a symmetric positive definit matrix, which is not the case for integral equations featuring the double layer operator. Second, the properties of the system matrix depend heavily on the involved basis as well. Before we focus on wavelet bases in the next chapter, we first give a short overview on fast boundary element methods in general.

## 1.4 Fast Boundary Element Methods

Besides the wavelet matrix compression there are numerous different approaches to treat the kernel function $k(\boldsymbol{x}, \boldsymbol{y})$, which appears under the integral, some of which we will briefly

mention here.

Considering a basis $\phi_j$ for the spaces $V_j$, which live on a single scale of spatial resolution, results in a full system matrix. The complexity to assemble and solve the system of linear equations is then at least $\mathcal{O}(N^2)$, where $N$ denotes the number of degrees of freedom. Another issue is that the involved matrix can become more and more ill-conditioned as the number of degrees of freedom increases. These two obstacles have led to a lot of progress for mathematical methods on reducing the complexity to $\mathcal{O}(N \log(N))$, or even to $\mathcal{O}(N)$, some of which we will address here.

One of the key ideas for fast boundary element methods lies in the fact that, for sufficiently smooth kernel functions and if the distance of the points $\boldsymbol{x}$ and $\boldsymbol{y}$ is large enough, an approximation of the kernel function

$$k(\boldsymbol{x}, \boldsymbol{y}) \approx \tilde{k}_r(\boldsymbol{x}, \boldsymbol{y}) \coloneqq \sum_{l=1}^{r} g_l(\boldsymbol{x}) f_l(\boldsymbol{y}), \tag{1.37}$$

separating the two variables, may be used. This so-called degenerate kernel approximation allows us to separate the double integral in (1.36), under which the kernel function appears, into single integrals, featuring the $\boldsymbol{x}$ and $\boldsymbol{y}$ part separately.

Since (1.37) is applicable only if $\|\boldsymbol{x}-\boldsymbol{y}\|$ is large enough, one has to introduce an appropriate matrix partitioning as it is for example done for hierachical matrices [52, 88]. Suppose we have given an index set $\mathcal{I}$ of cardinality $N$ and our set of basis functions $\{\phi_i\}_{i\in\mathcal{I}}$. With these ingredients, we can build a so-called cluster tree $\mathcal{T}_\mathcal{I}$ (the nodes of this tree are called clusters), which is generated by subdividing the index set $\mathcal{I}$ into disjoint non-empty subsets $\iota$, $\iota'$ with $\mathcal{I} = \iota \cup \iota'$. The subdivision is repeated until the cluster has less than a prescribed number of indices. The leafs of this tree form then a partition of the original set $\mathcal{I}$. With this tree at hand, we form a block-cluster tree $\mathcal{T}_{\mathcal{I}\times\mathcal{I}}$ by combining clusters of the cluster

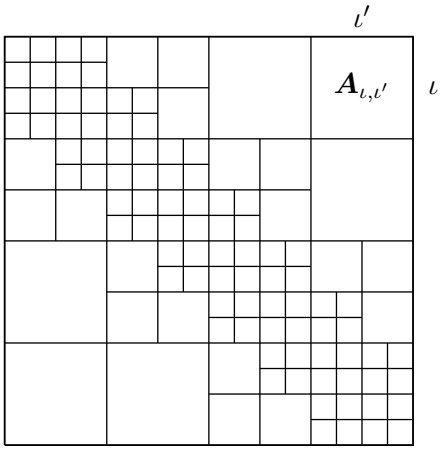



Figure 1.1: Structure of the block-cluster tree

tree. As we do this, we use, starting from the root, a recursive admissibility condition to check which of these blocks $\boldsymbol{A}_{\iota,\iota'} \in \mathbb{R}^{|\iota|\times|\iota'|}$ qualify for a low-rank approximation, i.e. by a matrix of rank $r$.

Using the local representation (1.37) for the matrix entries in (1.36) gives an approximation to the matrix block

$$\boldsymbol{A}_{\iota,\iota'} \approx \boldsymbol{L}\boldsymbol{R}^t \tag{1.38}$$

with matrices $\boldsymbol{L} \in \mathbb{R}^{|\iota|\times r}$ and $\boldsymbol{R} \in \mathbb{R}^{|\iota'|\times r}$ given by

$$\boldsymbol{L}_{i,l} \coloneqq \langle g_l, \phi_i \rangle, \quad \boldsymbol{R}_{i',l} \coloneqq \langle f_l, \phi_{i'} \rangle.$$

Blocks of dimension $|\iota| \times |\iota'|$ can thus be stored in $\mathcal{O}(r(|\iota| + |\iota'|))$ instead of $\mathcal{O}(|\iota| \cdot |\iota'|)$. In the following we mention several methods to obtain the degenerated kernel expansion (1.37).

The fast multipole method, see [50, 79] for example, aims at rewriting the involved kernel function into a polynomial expansion by means of spherical harmonics. Another approach is provided by the so-called panel clustering by Hackbusch and Novak [53]. Here, the kernel is approximated by through a Taylor expansion, whereas [10] approximates the kernel function by polynomial interpolation.

While the aforementioned methods aim at approximating the involved kernel function by an explicit degenerated kernel approximation, the idea of the adaptive cross approximation [2, 4] is a different one. Namely, it rather uses a few exact entries of the original matrix for constructing the low-rank approximation (1.38) directly. The idea on how to determine which values should be used for this procedure is explained in e.g. [3]. The storage requirements for the block $\boldsymbol{A}_{\iota,\iota'} \in \mathbb{R}^{|\iota|\times|\iota'|}$ are then again $\mathcal{O}(r(|\iota| + |\iota'|))$.

All of the mentioned schemes realise the desired accuracy in complexity $\mathcal{O}(N \log(N))$, where $N$ denotes the number of degrees of freedom. The complexity can be further reduced to $\mathcal{O}(N)$ if $\mathcal{H}^2$-matrices are used. They are a special case of $\mathcal{H}$-matrices which exploit a nested hierarchy between different levels. We only mention [9] and the references therein. However, we are not aware of an algorithm using adaptive methods for $\mathcal{H}$-matrices or $\mathcal{H}^2$-matrices.

We will, in this thesis, consider basis functions with special properties to deal with the appearing problem of a full system matrix. These basis functions are called wavelets and their main feature is that the coefficients of the system matrix in the wavelet basis decay drastically as we move away from the singular parts. This allows us to neglect matrix entries without losing over-all accuracy [7]. Using only this property already reduces the complexity to $\mathcal{O}(N \log(N))$, with $N$ denoting again the number of unknowns. Then, using a second estimate for neglecting even more matrix entries [84] results in linear complexity $\mathcal{O}(N)$.

# 2

# Wavelets

In this chapter, we will present the wavelet bases under consideration. We have already mentioned that the structure of the system matrix after discretisation depends heavily on the choice of the basis. Choosing a single-scale basis for discretisation will result in a dense system matrix. However, choosing a wavelet basis will result in a quasi-sparse matrix, as the representation of the operator in wavelet coordinates contains a large amount of very small entries, which can be neglected without compromising the over-all accuracy.

The structure of this chapter is as follows. In the first section, we will introduce the primal and dual single-scale bases and associated wavelets in one spatial dimension. We will then define the wavelets on the unit interval, where we will not go too much into detail on their construction, since this would involve an extensive discussion on biorthogonalisation techniques and stable completions, see e.g. [12, 30, 32] for more details. After these functions are introduced, we continue to the wavelets on the unit square, built by taking the tensor product of wavelets on the unit interval. Thereby, we will introduce a second form of representing the wavelet basis, for which the functions are optimised with respect to their support. These are the functions which are finally used in our implementation. Subsequently, we focus on how to represent surfaces of three-dimensional domains, where we use mappings from the unit square to the according patches on the surface of the geometry under consideration. These mappings are applied in the last section to construct piecewise constant wavelets on the surface. For the discussion of globally continuous wavelet bases on surfaces and for higher order ansatz functions, we refer the reader to [18, 30, 54, 60].

## 2.1 Primal and Dual Scaling Functions on $\mathbb{R}$

This section is concerned with introducing the primal and dual basis functions, called scaling functions, as well as the concept of a multiresolution analysis on $\mathbb{R}$. We restrict ourselves to piecewise constant scaling functions to construct the underlying multiresolution analysis, since we only consider piecewise constant wavelets in this thesis and for our implementation. For the construction of spline wavelets of arbitrary order $m$, see e.g. [18, 30, 54].

Let us start by defining the scaling function $\phi^{\mathbb{R}}(t)$ as the characteristic function of the interval $I$ (being the B-spline of order $m = 1$). Without loss of generality, we can assume that $I = [0, 1]$. This function is compactly supported

$$\phi^{\mathbb{R}}(t) \neq 0 \Leftrightarrow t \in I$$

and normalised

$$\int_{\mathbb{R}} \phi^{\mathbb{R}}(t)\mathrm{d}t = 1.$$

Our scaling functions are subject to a refinement relation, which means that they can be expressed as a combination $\phi^{\mathbb{R}}(t) = \phi^{\mathbb{R}}(2t-1) + \phi^{\mathbb{R}}(2t)$ of translated and dilated versions of themselves. By considering the dilates $\phi^{\mathbb{R}}(2^j t)$ with respect to an arbitrary refinement scale $j \in \mathbb{Z}$ and the translates $\phi^{\mathbb{R}}(t-k)$ with arbitrary $k \in \mathbb{Z}$, we can define the set of functions

$$\phi^{\mathbb{R}}_{j,k}(t) = 2^{j/2}\phi^{\mathbb{R}}(2^j t - k). \tag{2.1}$$

The factor $2^{j/2}$ is introduced for normalisation, leading us to $L^2$-normalised functions

$$\phi^{\mathbb{R}}_{j,k}(t) = \begin{cases} 2^{j/2}, & \text{if } t \in \left[2^{-j}k, 2^{-j}(k+1)\right), \\ 0, & \text{else.} \end{cases}$$

The integer $k \in \mathbb{Z}$ which determines the translation, indicates at the same time the number of each basis function. Moreover, all of these functions have a compact support with respect to the scale $j$, that is $\mathrm{diam}(\mathrm{supp}(\phi^{\mathbb{R}}_{j,k})) = 2^{-j}$.

**Definition 7.** Let $\{t_i\}_{i\in\mathbb{Z}}$ be given vectors in a Hilbert space $V$ and $\overline{\mathrm{span}\{t_i\}} = V$. If every element $v \in V$ has a unique expansion $v = \sum_{i\in\mathbb{Z}} v_i t_i$ and if there exist universal constants $c_1$ and $c_2$ such that

$$c_1\|\boldsymbol{v}\|_{\ell^2(\mathbb{Z})} \leq \|v\|_{L^2(\mathbb{R})} \leq c_2\|\boldsymbol{v}\|_{\ell^2(\mathbb{Z})},$$

then $\{t_i\}_{i\in\mathbb{Z}}$ is called a Riesz basis. Here, with $\boldsymbol{v} = \{v_i\}_{i\in\mathbb{Z}}$ we denote the vector of coefficients and with $\ell^2(\mathbb{Z})$ the space of square summable sequences.

For each $j$, the collections $\mathbf{\Phi}_j^{\mathbb{R}} = \{\phi_{j,k}^{\mathbb{R}}\}_{k\in\mathbb{Z}}$, introduced in (2.1), form Riesz bases and they generate the spaces $V_j^{\mathbb{R}} = \overline{\operatorname{span}(\mathbf{\Phi}_j^{\mathbb{R}})}$. For the parameter $\tilde{m} \geq m$ (and $\tilde{m}+m$ even), there exists a scaling function $\tilde{\phi}^{\mathbb{R}}$ (with polynomial exactness $\tilde{m}$), being biorthogonal, meaning that

$$\int_{\mathbb{R}} \phi^{\mathbb{R}}(t-k)\,\tilde{\phi}^{\mathbb{R}}(t-k')\mathrm{d}t = \delta_{k,k'} \quad \text{for } k,k' \in \mathbb{Z}, \tag{2.2}$$

cf. [18]. Like before, we obtain the set of functions $\tilde{\phi}_{j,k}^{\mathbb{R}}(t) = 2^{j/2}\tilde{\phi}^{\mathbb{R}}(2^j t - k)$ by considering the translates and dilates. They are again compactly supported, normalised and refinable [25].

The collections $\tilde{\mathbf{\Phi}}_j^{\mathbb{R}} = \{\tilde{\phi}_{j,k}^{\mathbb{R}}\}_{k\in\mathbb{Z}}$ of the dual functions form again Riesz bases and generate the dual spaces $\tilde{V}_j^{\mathbb{R}} = \overline{\operatorname{span}(\tilde{\mathbf{\Phi}}_j^{\mathbb{R}})}$. The spaces $V_j^{\mathbb{R}}$ and $\tilde{V}_j^{\mathbb{R}}$ generated like this form a primal and dual multiresolution analysis in $L^2(\mathbb{R})$:

**Definition 8.** A multiresolution analysis of the space $L^2(\mathbb{R})$ consists of a sequence of nested spaces $\cdots \subset V_{-2}^{\mathbb{R}} \subset V_{-1}^{\mathbb{R}} \subset V_0^{\mathbb{R}} \subset V_1^{\mathbb{R}} \subset V_2^{\mathbb{R}} \subset \cdots$ which satisfy

$$\overline{\bigcup_{j\in\mathbb{Z}} V_j^{\mathbb{R}}} = L^2(\mathbb{R}) \text{ and } \bigcap_{j\in\mathbb{Z}} V_j^{\mathbb{R}} = \emptyset.$$

Furthermore, for a function $f \in L^2(\mathbb{R})$, it is required that

$$\begin{aligned} f(t) \in V_0^{\mathbb{R}} &\iff f(2^j t) \in V_j^{\mathbb{R}} \quad \text{for all } j \in \mathbb{Z}, \\ f(t) \in V_0^{\mathbb{R}} &\iff f(t-k) \in V_0^{\mathbb{R}} \quad \text{for all } k \in \mathbb{Z}. \end{aligned} \tag{2.3}$$

## 2.2 Wavelet Bases on $\mathbb{R}$

Suppose, we have chosen a basis in $V_j^{\mathbb{R}}$. If we decide that this level of refinement does not suffice, we have to choose a completely new basis for $V_{j+1}^{\mathbb{R}}$, throwing away the old one. Furthermore, representing nonlocal operators using a single-scale basis leads to dense system matrices. The idea of wavelets is to keep track of the increment of information between two subsequent spaces $V_j^{\mathbb{R}}$ and $V_{j+1}^{\mathbb{R}}$.

Suppose, we have given the primal and dual bases $\mathbf{\Phi}_j^{\mathbb{R}}$ and $\tilde{\mathbf{\Phi}}_j^{\mathbb{R}}$ of the spaces $V_j^{\mathbb{R}}$ and $\tilde{V}_j^{\mathbb{R}}$, respectively. The goal is to find collections of functions $\{\psi_{j,k}^{\mathbb{R}}\}_{k\in\mathbb{Z}}$ and $\{\tilde{\psi}_{j,k}^{\mathbb{R}}\}_{k\in\mathbb{Z}}$, each set forming the complement spaces $W_j^{\mathbb{R}}$ and $\tilde{W}_j^{\mathbb{R}}$, such that

$$V_{j+1}^{\mathbb{R}} = V_j^{\mathbb{R}} \oplus W_{j+1}^{\mathbb{R}} \text{ and } \tilde{V}_{j+1}^{\mathbb{R}} = \tilde{V}_j^{\mathbb{R}} \oplus \tilde{W}_{j+1}^{\mathbb{R}}$$

hold for each $j \in \mathbb{Z}$. By using this property recursively, we get the multiscale decomposition

$$\overline{\bigoplus_{j\in\mathbb{Z}} W_j^{\mathbb{R}}} = \overline{\bigoplus_{j\in\mathbb{Z}} \tilde{W}_j^{\mathbb{R}}} = L^2(\mathbb{R}).$$

Aside from forming Riesz bases, the collections $\{\psi_{j,k}^{\mathbb{R}}\}_{k\in\mathbb{Z}}$ and $\{\tilde{\psi}_{j,k}^{\mathbb{R}}\}_{k\in\mathbb{Z}}$ shall fulfil the orthogonality relation

$$\int_{\mathbb{R}} \psi^{\mathbb{R}}(t-k)\,\tilde{\psi}^{\mathbb{R}}(t-k')\,\mathrm{d}t = \delta_{k,k'} \quad \text{for } k,k' \in \mathbb{Z}. \tag{2.4}$$

To find the sets $\{\psi_{j,k}^{\mathbb{R}}\}$ and $\{\tilde{\psi}_{j,k}^{\mathbb{R}}\}$, we use appropriate linear combinations of the functions $\phi^{\mathbb{R}}$ and $\tilde{\phi}^{\mathbb{R}}$ to construct functions $\psi^{\mathbb{R}}$ and $\tilde{\psi}^{\mathbb{R}}$ (see Section 2.3 for details). Involving their translates $\psi^{\mathbb{R}}(t-k)$ and dilates $\psi^{\mathbb{R}}(2^j t)$, we define the sets

$$\psi_{j,k}^{\mathbb{R}}(t) = 2^{j/2}\psi^{\mathbb{R}}(2^j t - k), \quad \tilde{\psi}_{j,k}^{\mathbb{R}}(t) = 2^{j/2}\tilde{\psi}^{\mathbb{R}}(2^j t - k).$$

Again, the factor $2^{j/2}$ is introduced due to normalisation, leading to $L^2$-normalised functions. According to [18], these functions fulfil (2.4) if equation (2.2) holds. Finally, the collections $\cup_{j\in\mathbb{Z}}\mathbf{\Psi}_j^{\mathbb{R}}$ and $\cup_{j\in\mathbb{Z}}\tilde{\mathbf{\Psi}}_j^{\mathbb{R}}$ form Riesz bases in $L^2(\mathbb{R})$ [18].

Since we have a biorthogonal basis, we have some useful relations between different spaces, namely $\tilde{W}_{j+1}^{\mathbb{R}} \perp V_j^{\mathbb{R}}$, $W_{j+1}^{\mathbb{R}} \perp \tilde{V}_j^{\mathbb{R}}$ and $W_j^{\mathbb{R}} \perp \tilde{W}_{j'}^{\mathbb{R}}$ for $j \neq j'$. Also, the spaces $W_j^{\mathbb{R}}$ and $\tilde{W}_j^{\mathbb{R}}$ inherit a similar property as the single-scale spaces in equation (2.3). If the set $\tilde{\phi}_{j,k}^{\mathbb{R}}$ of dual scaling functions is exact of order $\tilde{m}$, then the property $W_{j+1}^{\mathbb{R}} \perp \tilde{V}_j^{\mathbb{R}}$, together with equation (2.2) implies the so-called vanishing moment property for the primal wavelets

$$\int_{\mathbb{R}} t^r\,\psi_{j,k}^{\mathbb{R}}(t)\,\mathrm{d}t = 0 \quad \text{for } r < \tilde{m}. \tag{2.5}$$

Analogously, we remark that the dual wavelets have $m$ vanishing moments. Specifically we use wavelets with $(m,\tilde{m}) = (1,1)$ vanishing moments (Haar wavelets) and such with $(m,\tilde{m}) = (1,3)$ vanishing moments. In the upcoming section, we will see a specific wavelet construction which uses a combination of only finitely many single-scale functions $\phi^{\mathbb{R}}$. Since all of these have compact support, the constructed wavelets too have compact support with respect to the refinement scale $j$.

Let $\gamma$ and $\tilde{\gamma}$ ($\gamma,\tilde{\gamma} > 0$) denote the regularity of the basis functions $\phi^{\mathbb{R}}$ and $\tilde{\phi}^{\mathbb{R}}$, that is

$$\gamma = \sup\{s \in \mathbb{R} \colon \phi^{\mathbb{R}}(t) \in H^s(\mathbb{R})\} \quad \text{and} \quad \tilde{\gamma} = \sup\{s \in \mathbb{R} \colon \tilde{\phi}^{\mathbb{R}}(t) \in H^s(\mathbb{R})\}.$$

As our primal multiresolution is based on smoothest splines, the regularity of the ansatz functions is $\gamma = m - 1/2 = 1/2$.

For parameters $s_1 \leq s_2 \leq m = 1$ with $q < \gamma$ we have the following approximation estimate

$$\inf_{v_j \in V_j^{\mathbb{R}}} \|u - v_j\|_{H^{s_1}(\mathbb{R})} \lesssim 2^{j(s_1-s_2)}\|u\|_{H^{s_2}(\mathbb{R})} \quad \text{for } u \in H^{s_2}(\mathbb{R}) \quad \text{(Jackson)}. \tag{2.6}$$

Also there holds the inverse estimate

$$\|v_j\|_{H^{s_2}(\mathbb{R})} \lesssim 2^{j(s_2-s_1)}\|v_j\|_{H^{s_1}(\mathbb{R})} \quad \text{for all } v_j \in V_j^{\mathbb{R}} \quad \text{(Bernstein)} \tag{2.7}$$

for $s_1 \leq s_2 < \gamma$, see e.g. [30, 60]. Analogous estimates hold for the dual multiresolution analysis.

With the Jackson and Bernstein estimate holding, we have the norm equivalences [29,43,66]

$$\|v\|^2_{H^s(\mathbb{R})} \sim \begin{cases} \sum_{j,k\in\mathbb{Z}} 2^{js}|(v,\tilde{\psi}_{j,k})|^2, & s \in (-\tilde{\gamma},\gamma), \\ \sum_{j,k\in\mathbb{Z}} 2^{js}|(v,\psi_{j,k})|^2, & s \in (-\gamma,\tilde{\gamma}). \end{cases}$$

## 2.3 Wavelet Bases on the Interval

After having introduced the idea of the primal and dual multiresolution analysis and the wavelet bases on $\mathbb{R}$, we now turn to the bases on the unit interval $[0,1]$. Other than on $\mathbb{R}$, where any translate of a wavelet is acceptable as basis function, this may not be the case on the unit interval as we arrive at the interval boundary. This is why the construction of a suitable primal and dual wavelet basis is tedious. Biorthogonalisation techniques are needed and appropriate functions, which form a stable completion, must be found. In this section, we specify two wavelet bases, which will later be used for the implementation, and state the properties that they fulfil. All the left out theoretical background on biorthogonalisation techniques or stable completions can be found in e.g. [12,30,54].

### 2.3.1 Scaling Functions

Let us start by introducing the index set $\Delta_j^{[0,1]} := \{0,\dots,2^j-1\}$ of cardinality $2^j$. It contains the indices $k$ of the piecewise constant functions $\phi_{j,k}$ with respect to a uniform refinement of level $j$ on the unit interval $[0,1]$. This gives us the $L^2$-normalised basis of $V_j$

$$\phi_{j,k}^{[0,1]} = 2^{j/2}\chi_{[2^{-j}k,2^{-j}(k+1)]} \quad \text{with } k \in \Delta_j^{[0,1]} \tag{2.8}$$

for each refinement level $j$. Let us denote the set of single-scale functions on a refinement scale $j$ as $\mathbf{\Phi}_j^{[0,1]} = \{\phi_{j,k}^{[0,1]}\}_{k\in\Delta_j^{[0,1]}}$. These scaling functions form Riesz bases for the spaces $V_j^{[0,1]} = \operatorname{span}(\mathbf{\Phi}_j^{[0,1]})$, are refinable in accordance with

$$\phi_{j,k}^{[0,1]} = \frac{1}{\sqrt{2}}\left(\phi_{j+1,2k}^{[0,1]} + \phi_{j+1,2k+1}^{[0,1]}\right), \tag{2.9}$$

compactly supported and exact of order $m = 1$. One can also construct a set of dual scaling functions $\tilde{\phi}_{j,k}^{[0,1]}$, see [30, 54], such that they form a basis of the dual space $\tilde{V}_j^{[0,1]}$. Finally, the spaces $V_j^{[0,1]}$ and $\tilde{V}_j^{[0,1]}$ form a primal and dual multiresolution analysis in $L^2([0,1])$.

With these biorthogonal bases $\mathbf{\Phi}_j^{[0,1]}$ and $\tilde{\mathbf{\Phi}}_j^{[0,1]}$ at hand, the next step is to construct a wavelet bases. Setting $W_0^{[0,1]} := V_0^{[0,1]}$ and $\tilde{W}_0^{[0,1]} := \tilde{V}_0^{[0,1]}$, the spaces $W_j^{[0,1]}$ and $\tilde{W}_j^{[0,1]}$ are uniquely defined for all $j \geq 0$ via

$$V_{j+1}^{[0,1]} = V_j^{[0,1]} \oplus W_{j+1}^{[0,1]} \quad \text{and} \quad \tilde{V}_{j+1}^{[0,1]} = \tilde{V}_j^{[0,1]} \oplus \tilde{W}_{j+1}^{[0,1]},$$

where $W_{j+1}^{[0,1]} \perp \tilde{V}_j^{[0,1]}$ and $\tilde{W}_{j+1}^{[0,1]} \perp V_j^{[0,1]}$. Next, we will give two explicit examples of wavelet bases on the interval. These bases will build the foundation for the two-dimensional wavelet bases on the unit square, which we are using for our implementation.

### 2.3.2 Haar Basis

The first example are the Haar wavelets. Given the piecewise constant primal and dual basis functions on the interval $[0,1]$ and the index set $\Delta_j^{[0,1]}$, we consider the difference set $\Delta_j^{[0,1]} \setminus \Delta_{j-1}^{[0,1]} = \{2^{j-1}, \cdots, 2^j - 1\}$. We easily see that this set is equivalent to the set $\{0, \cdots, 2^{j-1} - 1\}$ by shifting the indices by the factor $2^{j-1}$. We will denote this shifted difference set by $\nabla_j^{[0,1]}$ for $j > 0$. As wavelet functions we can simply take the linear combination

$$\psi_{j,k}^{[0,1]} = \frac{1}{\sqrt{2}} \left( \phi_{j,2k}^{[0,1]} - \phi_{j,2k+1}^{[0,1]} \right) \quad \text{with } k \in \nabla_j^{[0,1]}$$

for the primal wavelet basis, as well as for the dual wavelet basis due to orthonormality, i.e.

$$\psi_{j,k}^{[0,1]} = \tilde{\psi}_{j,k}^{[0,1]}.$$

Hence, the primal and dual basis consists of the same functions everywhere, i.e. there is no special boundary modification. By $\mathbf{\Psi}_j^{[0,1]}$ we denote the collection

$$\mathbf{\Psi}_j^{[0,1]} = \bigcup_{j'>0}^{j} \psi_{j',k}^{[0.1]} \cup \mathbf{\Phi}_0^{[0,1]} \quad \text{with } k \in \nabla_{j'}^{[0,1]}.$$

Analogously we define the dual basis $\tilde{\mathbf{\Phi}}_j^{[0,1]}$. In Figure 2.1 we see the scaling function $\phi_{0,0}^{[0,1]}$, the first two refined functions $\phi_{1,0}^{[0,1]}$, $\phi_{1,1}^{[0,1]}$ and the Haar wavelet $\psi_{1,0}^{[0,1]}$ on the unit interval. The index set $\nabla_j^{[0,1]}$ contains the indices $k$ of the functions $\psi_{j,k}^{[0,1]}$ and $\tilde{\psi}_{j,k}^{[0,1]}$, forming the complementary spaces $W_j^{[0,1]}$ and $\tilde{W}_j^{[0,1]}$, which coincide here. The wavelet functions obtained in this way feature $(m, \tilde{m}) = (1,1)$ vanishing moments and the collection $\mathbf{\Psi}_j^{[0,1]}$ forms a Riesz basis (in this case it even forms an orthonormal basis) in $L^2([0,1])$. Also, the wavelets have the same regularity as the Haar wavelets on $\mathbb{R}$ and they are clearly compactly supported.

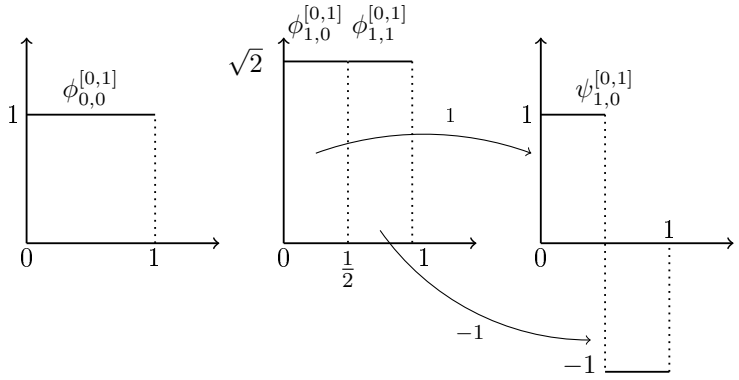



Figure 2.1: Scaling function, first two refined single-scale functions and Haar wavelet.

Furthermore, there holds the norm equivalence (recall that $\psi_{j,k}^{[0,1]} = \tilde{\psi}_{j,k}^{[0,1]}$)

$$\|v\|_{H^s([0,1])}^2 \sim \sum_{j\geq 0} \sum_{k\in\nabla_j^{[0,1]}} 2^{js} |(v, \psi_{j,k}^{[0,1]})|^2, \quad s \in \left(-\frac{1}{2}, \frac{1}{2}\right).$$

### 2.3.3 Wavelets With Three Vanishing Moments

For constructing the wavelets with three vanishing moments, we either have to start with a larger interval for defining the scaling functions, or start with a refinement scale which is larger than 0. This we have to do because the resulting wavelets have a larger support, requiring more singe-scale functions for their construction. Hence, suppose we have given the interval $[0, 1]$ with a large enough initial refinement. Particularly in our case, this means that these functions can be built by using single-scale functions on level $j = 3$. The primal scaling functions follow the same refinement relation (2.9) as for the construction of the Haar wavelets. For the dual scaling functions, we have the refinement relations cf. [54]

$$\tilde{\phi}_{j,0}^{[0,1]} = \frac{1}{\sqrt{2}} \left( \frac{11}{8}\tilde{\phi}_{j,0}^{[0,1]} + \frac{5}{8}\tilde{\phi}_{j,1}^{[0,1]} + \frac{1}{8}\tilde{\phi}_{j,2}^{[0,1]} - \frac{1}{8}\tilde{\phi}_{j,3}^{[0,1]} \right),$$

$$\tilde{\phi}_{j,1}^{[0,1]} = \frac{1}{\sqrt{2}} \left( -\frac{1}{2}\tilde{\phi}_{j,0}^{[0,1]} + \frac{1}{2}\tilde{\phi}_{j,1}^{[0,1]} + \tilde{\phi}_{j,2}^{[0,1]} + \tilde{\phi}_{j,3}^{[0,1]} + \frac{1}{8}\tilde{\phi}_{j,4}^{[0,1]} - \frac{1}{8}\tilde{\phi}_{j,5}^{[0,1]} \right),$$

$$\tilde{\phi}_{j,2}^{[0,1]} = \frac{1}{\sqrt{2}} \left( \frac{1}{8}\tilde{\phi}_{j,0}^{[0,1]} - \frac{1}{8}\tilde{\phi}_{j,1}^{[0,1]} - \frac{1}{8}\tilde{\phi}_{j,2}^{[0,1]} + \frac{1}{8}\tilde{\phi}_{j,3}^{[0,1]} + \tilde{\phi}_{j,4}^{[0,1]} + \tilde{\phi}_{j,5}^{[0,1]} + \frac{1}{8}\tilde{\phi}_{j,6}^{[0,1]} - \frac{1}{8}\tilde{\phi}_{j,7}^{[0,1]} \right).$$

These three equations represent the refinement relations for the boundary functions at the left interval boundary. Similar equations hold (modulo leading signs) for the right boundary functions $\tilde{\phi}_{j,2^{j-1}-1}$, $\tilde{\phi}_{j,2^{j-1}-2}$ and $\tilde{\phi}_{j,2^{j-1}-3}$ as well. For the interior scaling functions, we

have the refinement relation

$$\tilde{\phi}_{j,k}^{[0,1]} = \frac{1}{\sqrt{2}}\left(-\frac{1}{8}\tilde{\phi}_{j,2k-2}^{[0,1]} + \frac{1}{8}\tilde{\phi}_{j,2k-1}^{[0,1]} + \tilde{\phi}_{j,2k}^{[0,1]} + \tilde{\phi}_{j,2k+1}^{[0,1]} + \frac{1}{8}\tilde{\phi}_{j,2k+2}^{[0,1]} - \frac{1}{8}\tilde{\phi}_{j,2k+3}^{[0,1]}\right),$$

for $k \in \{3, \dots, 2^{j-1} - 4\}$. Choosing the Haar basis as stable completion, see e.g. [30, 54], we finally end up with the following variations of wavelets

$$\psi_{j,0}^{[0,1]} = \frac{1}{\sqrt{2}}\left(-\frac{5}{8}\phi_{j,0}^{[0,1]} + \frac{11}{8}\phi_{j,1}^{[0,1]} - \frac{1}{2}\phi_{j,2}^{[0,1]} - \frac{1}{2}\phi_{j,3}^{[0,1]} + \frac{1}{8}\phi_{j,4}^{[0,1]} + \frac{1}{8}\phi_{j,5}^{[0,1]}\right),$$

$$\psi_{j,2^{j-1}-1}^{[0,1]} = \frac{1}{\sqrt{2}}\left(+\frac{5}{8}\phi_{j,2^j-1}^{[0,1]} - \frac{11}{8}\phi_{j,2^j-2}^{[0,1]} + \frac{1}{2}\phi_{j,2^j-3}^{[0,1]} + \frac{1}{2}\phi_{j,2^j-4}^{[0,1]} - \frac{1}{8}\phi_{j,2^j-5}^{[0,1]} - \frac{1}{8}\phi_{j,2^j-6}^{[0,1]}\right),$$

$$\psi_{j,k}^{[0,1]} = \frac{1}{\sqrt{2}}\left(-\frac{1}{8}\phi_{j,2k-2}^{[0,1]} - \frac{1}{8}\phi_{j,2k-1}^{[0,1]} + \phi_{j,2k}^{[0,1]} - \phi_{j,2k+1}^{[0,1]} + \frac{1}{8}\phi_{j,2k+2}^{[0,1]} + \frac{1}{8}\phi_{j,2k+3}^{[0,1]}\right),$$

for $k \in \{1, \dots, 2^{j-1} - 2\}$. Note that we have only two wavelets (one at each boundary) which do not coincide with the interior functions, which are given by the first and second equation. Figure 2.2 shows us the three different (primal) wavelets on the interval.

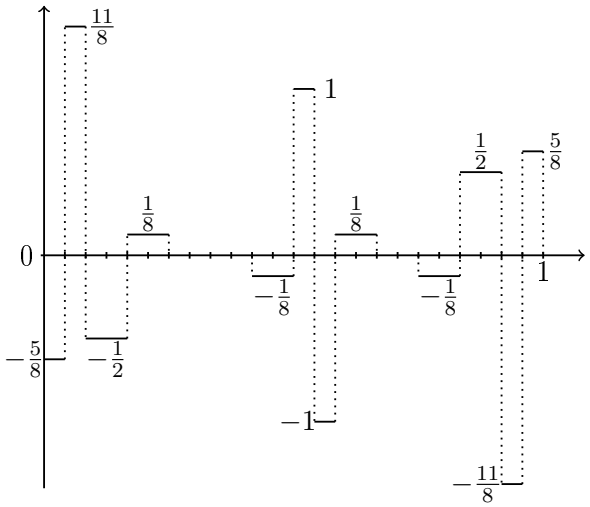



Figure 2.2: Wavelet with three vanishing moments with the left and right boundary modification.

The wavelets $\psi_{j,k}^{[0,1]}$ provide for all $k \in \nabla_j^{[0,1]}$ three vanishing moments, i.e.

$$\int_0^1 t^r \psi_{j,k}^{[0,1]}(t)\,\mathrm{d}t = 0 \quad \text{for } r < 3.$$

More precisely, it holds $(m, \tilde{m}) = (1, 3)$. Notice that all basis functions (primal and dual)

are compactly supported. The norm equivalences hold in the ranges

$$\|v\|^2_{H^s([0,1])} \sim \begin{cases} \sum_{j\geq 0}\sum_{k\in\nabla_j^{[0,1]}} 2^{js}\left|(v,\tilde{\psi}_{j,k}^{[0,1]})\right|^2, & s\in\left(-\tilde{\gamma},\tfrac{1}{2}\right), \\ \sum_{j\geq 0}\sum_{k\in\nabla_j^{[0,1]}} 2^{js}\left|(v,\psi_{j,k}^{[0,1]})\right|^2, & s\in\left(-\tfrac{1}{2},\tilde{\gamma}\right). \end{cases}$$

## 2.4 Wavelet Bases on the Unit Square

In this section, we will present the construction of wavelet bases on the unit square $\square = [0,1]\times[0,1]$. We make use of the introduced bases $\mathbf{\Phi}_j^{[0,1]}$ and $\tilde{\mathbf{\Phi}}_j^{[0,1]}$ of the spaces $V_j^{[0,1]}$ and $\tilde{V}_j^{[0,1]}$ and $\mathbf{\Psi}_j^{[0,1]}$ and $\tilde{\mathbf{\Psi}}_j^{[0,1]}$ on the interval. The general construction of wavelet bases on the $n$-dimensional unit cube can be found in e.g. [32, 54].

Let us start by defining the sets of scaling functions on the unit square. To this end, we introduce the vectors $\boldsymbol{t} := (t_1,t_2)$ denoting the coordinates in the square and $\boldsymbol{k} := (k_1,k_2)\in\Delta_j^\square := \Delta_j^{[0,1]}\times\Delta_j^{[0,1]}$ the pair of indices. We then define the set of primal and dual scaling functions on the square by taking the tensor product of the one-dimensional basis functions on the unit interval

$$\mathbf{\Phi}_j^\square = \mathbf{\Phi}_j^{[0,1]}\otimes\mathbf{\Phi}_j^{[0,1]}, \quad \tilde{\mathbf{\Phi}}_j^\square = \tilde{\mathbf{\Phi}}_j^{[0,1]}\otimes\tilde{\mathbf{\Phi}}_j^{[0,1]}. \tag{2.10}$$

This means that each basis function is defined by a product of one-dimensional basis functions

$$\phi_{j,\boldsymbol{k}}^\square(\boldsymbol{t}) = \phi_{j,k_1}^{[0,1]}(t_1)\cdot\phi_{j,k_2}^{[0,1]}(t_2), \quad \tilde{\phi}_{j,\boldsymbol{k}}^\square(\boldsymbol{t}) = \tilde{\phi}_{j,k_1}^{[0,1]}(t_1)\cdot\tilde{\phi}_{j,k_2}^{[0,1]}(t_2).$$

The sets $\mathbf{\Phi}_j^\square = \{\phi_{j,\mathbf{k}}^\square\}_{\boldsymbol{k}\in\Delta_j^\square}$ and $\tilde{\mathbf{\Phi}}_j^\square = \{\tilde{\phi}_{j,\mathbf{k}}^\square\}_{\boldsymbol{k}\in\Delta_j^\square}$ generate the spaces $V_j^\square = \operatorname{span}(\mathbf{\Phi}_j^\square)$ and $\tilde{V}_j^\square = \operatorname{span}(\tilde{\mathbf{\Phi}}_j^\square)$, respectively. The spaces $V_j^\square$ and $\tilde{V}_j^\square$ are nested, exact of order $m$ and $\tilde{m}$, respectively, and the union of all $V_j^\square$ and $\tilde{V}_j^\square$, $j\geq 0$, is dense in $L^2(\square)$. We again search functions $\psi^\square$ an $\tilde{\psi}^\square$ in the complement spaces $W_{j+1}^\square$ and $\tilde{W}_{j+1}^\square$ to enrich $V_j^\square$ and $\tilde{V}_j^\square$, respectively, in such a way that we get the next finer space

$$V_{j+1}^\square = V_j^\square\oplus W_{j+1}^\square \text{ and } \tilde{V}_{j+1}^\square = \tilde{V}_j^\square\oplus\tilde{W}_{j+1}^\square.$$

In particular, it holds $\tilde{W}_{j+1}^\square\perp V_j^\square$ and $W_{j+1}^\square\perp\tilde{V}_j^\square$. Nevertheless, it is not obvious how to define the associated wavelets. This is the point where a certain degree of freedom comes in and different wavelet bases can be chosen. We will explain the idea of how we can construct such bases, in particular, we use two specific examples of wavelet functions.

As introduced in Section 2.3, we use the Haar basis and the wavelets with three vanishing moments on the interval to build the tensor product wavelets. We shall not simply take ordinary tensor product wavelets, but construct wavelets which are optimised with respect

to their support. These have much smaller supports which leads to superior compression results. This is the main reason why we use them for our implementation.

### 2.4.1 Ordinary Tensor Product Wavelets

The first possibility to decompose the space $V_{j+1}^{\square}$ is according to

$$\begin{aligned} V_{j+1}^{\square} &= V_{j+1}^{[0,1]} \otimes V_{j+1}^{[0,1]} \\ &= \left(V_j^{[0,1]} \oplus W_{j+1}^{[0,1]}\right) \otimes \left(V_j^{[0,1]} \oplus W_{j+1}^{[0,1]}\right) \\ &= V_j^{\square} \oplus \left(W_{j+1}^{[0,1]} \otimes V_j^{[0,1]}\right) \oplus \left(V_j^{[0,1]} \otimes W_{j+1}^{[0,1]}\right) \oplus \left(W_{j+1}^{[0,1]} \otimes W_{j+1}^{[0,1]}\right). \end{aligned} \tag{2.11}$$

In the third equality, we used the rule $H_1 \otimes (H_2 \oplus H_3) = (H_1 \otimes H_2) \oplus (H_1 \otimes H_3)$. If we do not further transform this representation, we can take the tensor products of functions in the above defined spaces. We then end up with four different kinds of basis functions; the scaling functions from equation (2.10) together with three sorts of wavelets:

$$\begin{aligned} \psi_{j,\boldsymbol{k},1}^{\square}(\boldsymbol{t}) &= \psi_{j,k_1}^{[0,1]}(t_1) \cdot \phi_{j,k_2}^{[0,1]}(t_2) \quad \text{with } \boldsymbol{k} = (k_1, k_2) \in \nabla_{j,1}^{\square}, \\ \psi_{j,\boldsymbol{k},2}^{\square}(\boldsymbol{t}) &= \phi_{j,k_1}^{[0,1]}(t_1) \cdot \psi_{j,k_2}^{[0,1]}(t_2) \quad \text{with } \boldsymbol{k} = (k_1, k_2) \in \nabla_{j,2}^{\square}, \\ \psi_{j,\boldsymbol{k},3}^{\square}(\boldsymbol{t}) &= \psi_{j,k_1}^{[0,1]}(t_1) \cdot \psi_{j,k_2}^{[0,1]}(t_2) \quad \text{with } \boldsymbol{k} = (k_1, k_2) \in \nabla_{j,3}^{\square}. \end{aligned}$$

The index sets $\nabla_{j,\{1,2,3\}}^{\square}$ are all representing the set of basis functions. Each set is composed of the tensor product of two one-dimensional index sets on $[0,1]$ and can take different shapes. In particular, we have the sets $\nabla_{j,1}^{\square} = \nabla_j^{[0,1]} \times \Delta_j^{[0,1]}$, $\nabla_{j,2}^{\square} = \Delta_j^{[0,1]} \times \nabla_j^{[0,1]}$ and $\nabla_{j,3}^{\square} = \nabla_j^{[0,1]} \times \nabla_j^{[0,1]}$.

### 2.4.2 Wavelets With Optimised Support

It is possible to arrive at a different representation of (2.11) by a further transformation of the last two terms. We keep the first two addends $V_j^{\square}$ and $W_{j+1}^{[0,1]} \otimes V_j^{[0,1]}$ and use the relation $V_j^{[0,1]} \oplus W_{j+1}^{[0,1]} = V_{j+1}^{[0,1]}$ to obtain the following decomposition

$$V_{j+1}^{\square} = V_j^{\square} \oplus \left(W_{j+1}^{[0,1]} \otimes V_j^{[0,1]}\right) \oplus \left(V_{j+1}^{[0,1]} \otimes W_{j+1}^{[0,1]}\right).$$

The advantage of this representation is that it produces wavelets with a smaller support, since the involved single-scale space in the third addend is $V_{j+1}^{[0,1]}$ instead of $V_j^{[0,1]}$. This may seem insignificant from the analytical point of view, but it is of great importance for speeding up the computations since the matrix can be compressed more efficiently. We

have now two sorts of wavelets instead of three, namely

$$\psi^{\square}_{j,\boldsymbol{k},1}(\boldsymbol{t}) = \psi^{[0,1]}_{j,k_1}(t_1) \cdot \phi^{[0,1]}_{j,k_2}(t_2) \qquad \text{with } \boldsymbol{k} = (k_1, k_2) \in \nabla^{\square}_{j,1} = \nabla^{[0,1]}_j \times \Delta^{[0,1]}_j,$$
$$\psi^{\square}_{j,\boldsymbol{k},2}(\boldsymbol{t}) = \phi^{[0,1]}_{j+1,k_1}(t_1) \cdot \psi^{[0,1]}_{j,k_2}(t_2) \qquad \text{with } \boldsymbol{k} = (k_1, k_2) \in \nabla^{\square}_{j,2} = \Delta^{[0,1]}_{j+1} \times \nabla^{[0,1]}_j,$$

and likewise for the dual basis. Note that the first index set $\nabla^{[0,1]}_{j,1}$ is the same as before, the other set $\nabla^{[0,1]}_{j,2}$, however, is different.

We form the collections $\boldsymbol{\Psi}^{\square}_{j,\iota} = \{\psi^{\square}_{j,\boldsymbol{k},\iota}\}$ and $\tilde{\boldsymbol{\Psi}}^{\square}_{j,\iota} = \{\tilde{\psi}^{\square}_{j,\boldsymbol{k},\iota}\}$, respectively, with respect to the different index sets $\nabla^{\square}_{j,\iota}$ for $\iota = \{1,2\}$ for the optimised wavelets and $\iota = \{1,2,3\}$ for the ordinary tensor product wavelets. Ultimately, we have the set of primal wavelets $\boldsymbol{\Psi}^{\square}_j = \bigcup_\iota \boldsymbol{\Psi}^{\square}_{j,\iota}$ and dual wavelets $\tilde{\boldsymbol{\Psi}}^{\square}_j = \bigcup_\iota \tilde{\boldsymbol{\Psi}}^{\square}_{j,\iota}$. These sets span the complement spaces $W^{\square}_j = \operatorname{span}(\boldsymbol{\Psi}^{\square}_j)$ and $\tilde{W}^{\square}_j = \operatorname{span}(\tilde{\boldsymbol{\Psi}}^{\square}_j)$. Both mentioned wavelet bases feature $(m, \tilde{m}) = (1,1)$ and $(m, \tilde{m}) = (1,3)$ vanishing moments, respectively, similar to their one-dimensional counterparts on the unit interval which were used for their construction.

## 2.5 Wavelet Bases on the Surface

We will close this section by introducing wavelets on surfaces. To this end, we will first describe how we represent surfaces. Let $\Omega \subset \mathbb{R}^3$ be a bounded and simply connected domain with boundary $\Gamma \coloneqq \partial\Omega$. The surface of the geometry is divided into smooth, four-sided patches $\Gamma_i$, meaning that the surface $\Gamma$ is represented by the union

$$\Gamma = \bigcup_{i=1}^{M} \Gamma_i, \quad \Gamma_i = \boldsymbol{\gamma}_i(\square), \quad i = 1, \ldots, M$$

with $\boldsymbol{\gamma}_i \colon \square \to \Gamma_i$ being smooth diffeomorphisms from the unit square $\square = [0,1]^2$ to the patch $\Gamma_i$. Thereby, we assume that the intersection $\Gamma_i \cap \Gamma_{i'}$, $i \neq i'$, of two different patches is either empty, a common edge or a common vertex. Besides that, the mappings $\boldsymbol{\gamma}_i$ need to fulfil a certain conformity condition so that we will obtain a regular mesh. For each $\boldsymbol{x}$ in the intersection of two patches $\Gamma_i \cap \Gamma_{i'}$, there exists an affine map $\Xi \colon \square \to \square$ such that $\boldsymbol{x} = \boldsymbol{\gamma}_i(\boldsymbol{t}) = (\boldsymbol{\gamma}_i \circ \Xi)(\boldsymbol{t})$. Refinement of the surface is then automatically introduced by a refinement of the unit square. More precisely, a uniform mesh of level $j$ on $\Gamma$ is obtained by subdividing the square into $4^j$ portions (called elements) of the form

$$\square_{j,\boldsymbol{k}} \coloneqq [2^{-j}(k_1, k_1 + 1)] \times [2^{-j}(k_2, k_2 + 1)] \subseteq \square,$$

with $\boldsymbol{k} = (k_1, k_2)$ again indicating the location. When we assume that the surface is refined uniformly, this generates $M4^j$ elements $\Gamma_{i,j,\boldsymbol{k}}$ on the surface with $\Gamma_{i,j,\boldsymbol{k}} \coloneqq \boldsymbol{\gamma}_i(\square_{j,\boldsymbol{k}})$. In particular, we have $\Gamma_{i,0,(0,0)} = \Gamma_i$. Figure 2.3 shows two different elements and how they are represented on the unit square. Note that this way of representing the surface of a geometry is well known in Computer Aided Geometric Design. In particular, it is the

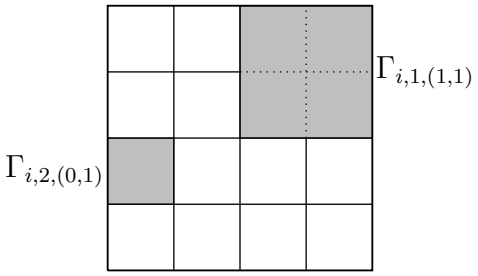



Figure 2.3: Element $\Gamma_{i,1,(1,1)}$ and element $\Gamma_{i,2,(0,1)}$.

topic of recent studies in isogeometric analysis [64].

Consider a surface $\Gamma$, which is represented in the described way. Suppose we have at hand the bases on the unit square. We can then construct the bases on the surface $\Gamma$, by using these mappings to represent scaling functions and wavelets on the surface in the following way, where $t \coloneqq \boldsymbol{\gamma}_i^{-1}(\boldsymbol{x})$:

$$\begin{aligned} \phi_{j,\boldsymbol{k}}^{\Gamma_i}(\boldsymbol{x}) &= \phi_{j,\boldsymbol{k}}^{\square}(\boldsymbol{t}) \qquad \text{and} \quad \psi_{j,\boldsymbol{k},\{1,2,3\}}^{\Gamma_i}(\boldsymbol{x}) = \psi_{j,\boldsymbol{k},\{1,2,3\}}^{\square}(\boldsymbol{t}) \\ \tilde{\phi}_{j,\boldsymbol{k}}^{\Gamma_i}(\boldsymbol{x}) &= \tilde{\phi}_{j,\boldsymbol{k}}^{\square}(\boldsymbol{t})/\mu_i(\boldsymbol{t}) \quad \text{and} \quad \tilde{\psi}_{j,\boldsymbol{k},\{1,2,3\}}^{\Gamma_i}(\boldsymbol{x}) = \tilde{\psi}_{j,\boldsymbol{k},\{1,2,3\}}^{\square}(\boldsymbol{t})/\mu_i(\boldsymbol{t}) \end{aligned} \tag{2.12}$$

for all $\boldsymbol{x} \in \Gamma_i$. Thereby, the expression $\mu_i$ stands for the surface measure which is defined as

$$\mu_i(\boldsymbol{t}) \;\coloneqq\; \left\| \frac{\partial \boldsymbol{\gamma}_i(\boldsymbol{t})}{\partial t_1} \times \frac{\partial \boldsymbol{\gamma}_i(\boldsymbol{t})}{\partial t_2} \right\|. \tag{2.13}$$

Note that it holds $\mu_i(\boldsymbol{t}) > 0$ for all $\boldsymbol{t} \in \square$ and $i = 1, \ldots, M$. We introduce the following notation. Given a wavelet $\psi_{j,\boldsymbol{k},\iota}^{\Gamma_i}$ of refinement level $j$ on the $i$-th patch $\Gamma_i$ with $\boldsymbol{k}$ indicating the location, then we will write $\psi_{\boldsymbol{\lambda}}^{\Gamma}$ with $\boldsymbol{\lambda} \coloneqq (i, j, \boldsymbol{k}, \iota)$. The last integer $\iota$ in this 4-tuple denotes the sort of wavelet. On one fixed scale of refinement $j$ we can define the collections $\boldsymbol{\Psi}_{\boldsymbol{\Lambda}(j)}^{\Gamma} = \{\psi_{\boldsymbol{\lambda}}^{\Gamma} \,|\, \boldsymbol{\lambda} \in \nabla_j^{\Gamma}\}$ and $\tilde{\boldsymbol{\Psi}}_{\boldsymbol{\Lambda}(j)}^{\Gamma} = \{\tilde{\psi}_{\boldsymbol{\lambda}}^{\Gamma} \,|\, \boldsymbol{\lambda} \in \nabla_j^{\Gamma}\}$, by collating the functions $\psi_{\boldsymbol{\lambda}}$ and $\tilde{\psi}_{\boldsymbol{\lambda}}$, respectively, for all $i$, $\boldsymbol{k}$, $\iota$. The index set $\nabla_j^{\Gamma}$ hereby generalises the aforementioned set $\nabla_j^{\square}$ and contains all possible variations of 4-tuples $(i, j, \boldsymbol{k}, \iota)$ for a fixed $j$. We introduce the following abbreviation for the level $j = |\boldsymbol{\lambda}|$. Finally, we get the complete sets of basis functions on the surface up to a certain fixed scale $J$ by the union of the bases of a refinement scale $j$, i.e.

$$\boldsymbol{\Psi}_{\boldsymbol{\Lambda}}^{\Gamma} = \bigcup_{j=0}^{J} \boldsymbol{\Psi}_{\boldsymbol{\Lambda}(j)}^{\Gamma} \quad \text{and} \quad \tilde{\boldsymbol{\Psi}}_{\boldsymbol{\Lambda}}^{\Gamma} = \bigcup_{j=0}^{J} \tilde{\boldsymbol{\Psi}}_{\boldsymbol{\Lambda}(j)}^{\Gamma}.$$

According to [32], we have the norm equivalences

$$\|v\|^2_{H^s(\Gamma)} \sim \begin{cases} \sum_{j\geq 0}\sum_{\boldsymbol{\lambda}\in\nabla_j^\Gamma} 2^{js}|(v,\tilde{\psi}_{\boldsymbol{\lambda}}^\Gamma)|^2, & s\in\left(-\min\left\{\frac{1}{2},\tilde{\gamma}\right\},\frac{1}{2}\right), \\ \sum_{j\geq 0}\sum_{\boldsymbol{\lambda}\in\nabla_j^\Gamma} 2^{js}|(v,\psi_{\boldsymbol{\lambda}}^\Gamma)|^2, & s\in\left(-\frac{1}{2},\min\left\{\frac{1}{2},\tilde{\gamma}\right\}\right), \end{cases}$$

which hold in the given ranges.

Finally, we have primal and dual biorthogonal Riesz bases in $L^2(\Gamma)$, generating the spaces $V_j^\Gamma$, $\tilde{V}_j^\Gamma$, $W_j^\Gamma$ and $\tilde{W}_j^\Gamma$ and forming a primal and dual multiresolution analysis. Furthermore, the primal wavelets have vanishing moments of order $\tilde{m}=1$ or $\tilde{m}=3$, dependent on the chosen basis, expressed by the more general counterpart to equation (2.5)

$$\int_\square \boldsymbol{t}^{\boldsymbol{\alpha}}\,\psi_{\boldsymbol{\lambda}}^\Gamma(\boldsymbol{\gamma}_i(\boldsymbol{t}))\,\mathrm{d}\boldsymbol{t}=0 \quad \text{for } |\boldsymbol{\alpha}|<\tilde{m} \tag{2.14}$$

with the multi-index $\boldsymbol{\alpha}=(\alpha_1,\alpha_2)$ and $|\boldsymbol{\alpha}|=\alpha_1+\alpha_2$. The dual wavelets have always vanishing moments of order $m=1$.

We choose an analogous ansatz as in Chapter 1 to represent the solution $\rho$ by a linear combination of basis functions, only this time we do so with respect to the other basis $\boldsymbol{\Psi}_{\boldsymbol{\Lambda}}^\Gamma=(\psi_{\boldsymbol{\lambda}}^\Gamma)_{\boldsymbol{\lambda}\in\Lambda}$, i.e.

$$\rho=\boldsymbol{\Psi}_{\boldsymbol{\Lambda}}^\Gamma\boldsymbol{\rho}^{\boldsymbol{\Psi}_{\boldsymbol{\Lambda}}^\Gamma}=\sum_{\boldsymbol{\lambda}}\rho_{\boldsymbol{\lambda}}^{\boldsymbol{\Psi}_{\boldsymbol{\Lambda}}^\Gamma}\psi_{\boldsymbol{\lambda}}. \tag{2.15}$$

This leads again to a system of linear equations

$$\boldsymbol{A}^{\boldsymbol{\Psi}_{\boldsymbol{\Lambda}}^\Gamma}\boldsymbol{\rho}^{\boldsymbol{\Psi}_{\boldsymbol{\Lambda}}^\Gamma}=\boldsymbol{f}^{\boldsymbol{\Psi}_{\boldsymbol{\Lambda}}^\Gamma} \tag{2.16}$$

with the entries $\boldsymbol{f}_{\boldsymbol{\lambda}}^{\boldsymbol{\Psi}_{\boldsymbol{\Lambda}}^\Gamma}$ of the right-hand side and the matrix $\boldsymbol{A}_{\boldsymbol{\lambda},\boldsymbol{\lambda}'}^{\boldsymbol{\Psi}_{\boldsymbol{\Lambda}}^\Gamma}$ given by

$$\boldsymbol{f}_{\boldsymbol{\lambda}}^{\boldsymbol{\Psi}_{\boldsymbol{\Lambda}}^\Gamma}=\langle f,\psi_{\boldsymbol{\lambda}}\rangle \quad \text{and} \quad \boldsymbol{A}_{\boldsymbol{\lambda},\boldsymbol{\lambda}'}^{\boldsymbol{\Psi}_{\boldsymbol{\Lambda}}^\Gamma}=\langle\mathcal{A}\psi_{\boldsymbol{\lambda}'},\psi_{\boldsymbol{\lambda}}\rangle. \tag{2.17}$$

In the following, we will omit the mentioning of the basis and the surface $\Gamma$ in the superscript as we will always use wavelet bases on surfaces. Whenever this is not the case, e.g. to work on the unit square for assembling the system matrix, we will emphasise it.

# 3

# Adaptive Wavelet Schemes

This chapter will focus on the theoretical framework for the adaptive wavelet algorithm and is structured as follows. In the first section, we consider the numerical treatment of boundary integral equations as introduced in Chapter 1 and motivate and explain the need for an adaptive approach. Under these aspects, the desire for more general function spaces, namely Besov spaces instead of Sobolev spaces, will become clear. In the second section, we introduce approximation spaces, weak $\ell^p$ spaces and, most of all, Besov spaces and establish the necessary analytical background. Afterwards, we motivate the concept of the best $N$-term (tree) approximation, which is essential in order to measure the error produced by the adaptive wavelet method. Finally, the theoretical framework for the key routines of the adaptive implementation, like COARSE, RHS and APPLY, will be presented. In this chapter we will focus on the theoretical aspects of these routines. We will state the purpose and the requirements for each of these blocks in order to produce an optimal algorithm and finally present some complexity estimates. Therein, we will encounter two important estimates for the entries of the involved boundary integral operator, which follow from the properties of the chosen wavelet basis. The use of both of these estimates is necessary to being able to compute an approximation to the boundary integral operator in optimal complexity.

## 3.1 Motivation and Background

Recall a boundary integral equation in general form as described in Chapter 1:

$$\mathcal{A}\rho = f \quad \text{in } \Gamma. \tag{3.1}$$

For many such problems, particularly for geometries with edges or if the right-hand side has singularities, the solution $\rho$ itself admits singularities and thus has limited Sobolev regularity [23, 33]. This raises two questions. First, is it necessary to uniformly refine in the whole domain of interest in order to obtain an accurate approximation to the solution? Second, are Sobolev spaces still the best function spaces to work with, and if not, what other function spaces would be appropriate? These questions and their discussion form the first part of this chapter.

The answer to the first question is rather straightforward, as approximating a solution featuring singularities can require a very strong local refinement in small parts of the geometry. Dependent on the extent of this refinement, it is neither possible nor computationally efficient to refine uniformly, even if large servers with extensive amounts of memory are involved. Even if storage is not the problem, the time needed to perform such computations may be too long. Therefore, an adaptive refinement is required for the approximation of the solution.

In Chapter 1, we introduced the representation of the solution $\rho$ by a linear combination of suitable functions $\rho = \sum_i \rho_i^{\boldsymbol{\Phi}} \phi_i$, the choice of which is specified in Chapter 2. Now, we aim at not choosing these functions in advance, but will rather use the estimated residual in each step in order to find the new wavelets to be added. Compared to finite element methods, we cannot compute the residuum exactly for boundary integral equations. Nevertheless, there exist reliable and efficient estimators for the residual [39] and also optimal convergence rates have been proven [41, 45]. However, the questions of computability and linear cost are not trivial and therefore they will form a prominent part of this chapter.

The algorithms presented in what follows are based on the developments of Cohen, Dahmen, DeVore, Gantumur, Harbrecht, Schneider and Stevenson and their work in e.g. [16, 17, 26–28, 46, 47]. The key ingredient throughout all of these articles is the equivalence of the underlying boundary integral equation in variational form to an infinite system of linear equations. To that end, let us introduce the infinite index set $\mathcal{J}_\star = \bigcup_{j=0}^{\infty} \nabla_j$ and assume that the wavelet functions are $H^s(\Gamma)$-normalised, i.e. $\|\psi_{\boldsymbol{\lambda}}\|_{H^s(\Gamma)} \sim 1$. The set $\boldsymbol{\Psi} = \{\psi_{\boldsymbol{\lambda}} \,|\, \boldsymbol{\lambda} \in \mathcal{J}_\star\}$ thus forms a Riesz basis in the energy space $H^s(\Gamma)$. The equivalent formulation of (3.1) into a well-posed infinite system of equations in $\ell^2(\mathcal{J}_\star)$ reads: Seek $\rho = \boldsymbol{\Psi}\boldsymbol{\rho} = \sum_{\boldsymbol{\lambda}\in\mathcal{J}_\star} \rho_{\boldsymbol{\lambda}} \psi_{\boldsymbol{\lambda}}$ such that

$$\boldsymbol{A}\boldsymbol{\rho} = \boldsymbol{f} \tag{3.2}$$

with $\boldsymbol{f} = \langle f, \psi_{\boldsymbol{\lambda}} \rangle_{\boldsymbol{\lambda}\in\mathcal{J}_\star}$ and $\boldsymbol{A} = \langle \mathcal{A}\psi_{\boldsymbol{\lambda}'}, \psi_{\boldsymbol{\lambda}} \rangle_{\boldsymbol{\lambda},\boldsymbol{\lambda}'\in\mathcal{J}_\star}$.

This is a convenient moment to emphasise the main difference between the adaptive wavelet methods and adaptive finite element methods. For standard finite element methods, at some point, one has to restrict the infinite dimensional problem to a finite dimensional setting. With this, there come stability conditions such as *inf-sup* conditions in order to ensure that the system of linear equations stays solvable and well-defined after the projection onto a finite dimensional subspace. For adaptive wavelet methods, we stay in the infinite-dimensional setting, which is why the aforementioned concepts are not needed

here. However, it is not possible to truly apply the infinite operator $\boldsymbol{A}$. Thus, there has to be a way of approximating the application of this infinite operator, especially one which guarantees optimal convergence. The restriction of the infinite wavelet basis to a finite subset $\mathcal{J} \subset \mathcal{J}_\star$ with cardinality $N$ can be realised by choosing the appropriate rows and columns from the underlying infinite dimensional matrix. To obtain a good approximation, we need a way to measure its error.

The next question is: How small can the approximation error become when choosing $N$ coefficients from the solution's expansion with respect to the wavelet basis? The idea of choosing the $N$ largest coefficients comes up naturally and is referred to as the best $N$-term approximation. Furthermore, for any target accuracy $\epsilon$, we want to approximate the solution with a computational complexity that stays proportional to the best $N$-term approximation of the solution.

In the case of uniform refinement, the building-blocks to approximate the solution $\boldsymbol{\rho}$ are taken from linear spaces. For smooth geometries and smooth right-hand sides, the solution is regular and uniform approximation works well. However, for non-smooth domains or singular right-hand sides, regularity of the solution is decreased and so may be the efficiency for the approximation [23, 33]. Both these articles deal with the question on which properties are desired for a function in order to determine its rate of approximation by using a nonlinear approximation method. The answer amounts to Besov spaces and measuring the smoothness of the solution therein. There are several ways to introduce Besov spaces. One possibility is to use moduli of smoothness and another is the use of wavelet coefficients. We are going to introduce Besov spaces in more detail in the following section, giving an insight on both mentioned perspectives. Throughout this section, we will draw upon the work of, e.g. [22–24, 33, 45] and give embedding results as well as convergence estimates for the error of the approximation to the solution.

## 3.2 Nonlinear Approximation

The question, this section will be concerned with, is basically the following: How good can we approximate complicated functions in a space $H$ by more simple functions from a sequence of spaces $\{V_i\}$, which is dense in $H$, and how to choose this sequence? Here, we do not want to restrict ourselves to linear approximation which is associated with uniform refinement, but allow the partition, which is achieved by refinement, to depend on the function to be approximated for a fixed number of unknowns. For this purpose, wavelets are exactly the right functions.

### 3.2.1 Best $N$-Term Approximation

Recall the equivalence of the boundary integral equation (3.1) to the infinite system of linear equations (3.2) in $\ell^2(\mathcal{J}_\star)$ with the infinite index $\mathcal{J}_\star$ and the $H^s(\Gamma)$-normalized wavelet basis

$\boldsymbol{\Psi}$. We restrict the infinite set to a finite subset $\mathcal{J} \subset \mathcal{J}_\star$ of cardinality $N$, ending up with the system of linear equations on the index set $\mathcal{J}$:

$$\boldsymbol{A}_\mathcal{J} \boldsymbol{\rho}_\mathcal{J} = \boldsymbol{f}_\mathcal{J}, \tag{3.3}$$

with $\boldsymbol{A}_\mathcal{J} = \langle \mathcal{A}\psi_{\boldsymbol{\lambda}'}, \psi_{\boldsymbol{\lambda}} \rangle_{\boldsymbol{\lambda},\boldsymbol{\lambda}' \in \mathcal{J}}$ and $\boldsymbol{f}_\mathcal{J} = \langle f, \psi_{\boldsymbol{\lambda}} \rangle_{\boldsymbol{\lambda} \in \mathcal{J}}$. We already raised the question of how small the approximation error becomes when choosing $N$ terms out of the infinite basis $\Psi$. One way of choosing coefficients is to choose them without a further constraint, which is referred to as the unconstrained $N$-term approximation. In contrast to this, we want to find the $N$ best coefficients, which means that the estimate

$$\|\boldsymbol{\rho} - \boldsymbol{\rho}_\mathcal{J}\|_{\ell^2(\mathcal{J}_\star)} \leq \|\boldsymbol{\rho} - \boldsymbol{\rho}_{\mathcal{J}'}\|_{\ell^2(\mathcal{J}_\star)}$$

shall hold for any other set $\mathcal{J}'$ with the same cardinality $N$. In other words, we minimize the approximation error in $\ell^2(\mathcal{J}_\star)$ of the infinite dimensional vector $\boldsymbol{\rho}$. The choice of these $N$ coefficients is referred to as the best $N$-term approximation. It comes natural to choose just the largest $N$ coefficients of $\boldsymbol{\rho}$.

### 3.2.2 Approximation Spaces

This subsection will be concerned with introducing the approximation spaces. We try to summarise the concept and most important properties explained in [33], where the complete discussion can be found. We choose the following setting, that is, a normed space $H$ with norm $\|\cdot\|_H$ and an unconditional basis $\boldsymbol{\chi} = \{\chi_i \,|\, i \in \mathcal{J}_\star\}$. Suppose, we want to approximate a function $f \in H$. If we consider linear approximation, we choose the linear space $H_N$ of dimension $N$ with its basis $\{\chi_i | \, i = 1, \ldots, N\}$, i.e. $H_N = \operatorname{span}\{\chi_i \,|\, i = 1, \ldots, N\}$. Using the ansatz $f_N = \sum_{i=1}^N c_i \chi_i$, the approximation error $E_N(f)$ between the function $f \in H$ and the approximant is measured by

$$E_N(f) = \inf_{f_N \in H_N} \|f - f_N\|_H.$$

For nonlinear approximation of a given $f \in H$, we use the $N$-term approximation and write the approximant $f_N$ as $f_N = \sum_{i \in \mathcal{J}} c_i \chi_i$ with $\mathcal{J}$ being any set of indices of maximal cardinality $N$, i.e. $\#\mathcal{J} \leq N$. Thereby, the linear space $H_N$ is replaced by the space $X_N$, containing all functions $f_N$ which can be expressed in the above manner. The sum of two elements from the space $X_N$ does not necessarily lie in $X_N$, explaining the term nonlinear approximation. The error for the $N$-term approximation then reads as

$$\sigma_{N,H}(f) = \inf_{f_N \in X_N} \|f - f_N\|_H.$$

We want to describe the set of functions that fulfil a certain rate of approximation, meaning

that

$$\sigma_{N,H}(f) = \mathcal{O}(N^{-s}), \quad N \to \infty. \tag{3.4}$$

All functions $f \in H$ fulfilling this will be grouped by one set which is called approximation space. For $s > 0$ and $0 < p \leq \infty$, the approximation space $\mathfrak{A}^s_{p,H}$ is defined as the set of all functions $f \in H$ such that

$$|f|_{\mathfrak{A}^s_{p,H}} := \begin{cases} \sum_{N\in\mathbb{N}} \left( [N^s \sigma_{N,H}(f)]^p \, \frac{1}{N} \right)^{1/p}, & 0 < p < \infty, \\ \sup_{N\in\mathbb{N}} N^s \sigma_{N,H}(f), & p = \infty, \end{cases} \tag{3.5}$$

is finite. Note that (3.4) describes the situation for $p = \infty$. Together with the norm $\|\cdot\|_H$, we get a quasi-norm (since the triangle inequality is not valid, but it holds $\|x + y\| \leq C(\|x\| + \|y\|)$) for the approximation space, i.e. $\|\cdot\|_{\mathfrak{A}^s_{p,H}} := \|\cdot\|_H + |\cdot|_{\mathfrak{A}^s_{p,H}}$.

For smaller values of $p$, fewer functions are contained in the space $\mathfrak{A}^s_{p,H}$, i.e.

$$\mathfrak{A}^s_{p,H} \subset \mathfrak{A}^s_{p',H}, \quad 0 < p < p' \leq \infty.$$

### 3.2.3 Sequence Spaces, Weak $\ell^p$ Spaces and Interpolation Spaces

Consider the sequence $\boldsymbol{x} = (x_N)_{N\in\mathbb{N}}$ with $x_N \in \mathbb{R}$ and denote by $\mathbb{R}^{\mathbb{N}}$ the set of all such sequences. Those sequences which satisfy

$$\sum_{N\in\mathbb{N}} |x_N|^p < \infty \quad \text{for } 0 < p < \infty \quad \text{or} \quad \sup_{N\in\mathbb{N}} |x_N| \quad \text{for } p = \infty$$

generate the sequence space $\ell^p(\mathbb{N})$. For $p \geq 1$, these spaces are Banach spaces with the norm induced by

$$\|\boldsymbol{x}\|_{\ell^p(\mathbb{N})} := \begin{cases} \left( \sum_{N\in\mathbb{N}} |x_N|^p \right)^{1/p}, & p < \infty, \\ \sup_{N\in\mathbb{N}} |x_N|, & p = \infty. \end{cases} \tag{3.6}$$

The same holds true if we consider the index set $\mathcal{J}_\star$ in place of $\mathbb{N}$, leading us to the sequence space $\ell^p(\mathcal{J}_\star)$. Furthermore, notice that it holds $\ell^p(\mathcal{J}_\star) \subset \ell^{p'}(\mathcal{J}_\star)$ for $p < p'$. For our purpose, the most important among these spaces is the space $\ell^2(\mathcal{J}_\star)$, being not only a Banach, but a Hilbert space.

Instead of considering just the sequence $\boldsymbol{x}$, we consider now the non-increasing rearrangement $(x^\star_N)_{N\in\mathbb{N}}$, meaning that $x^\star_{N+1} \leq x^\star_N$ for all $N \in \mathbb{N}$, and use it to define the weak

space $\ell_q^w$ by

$$\ell_q^w(\mathcal{J}_\star) := \left\{ \boldsymbol{x} \in \ell^q(\mathcal{J}_\star) \text{ such that } \sup_{N\in\mathbb{N}} N^{1/q}|x_N^\star| =: |\boldsymbol{x}|_{\ell_q^w} < \infty \right\}.$$

This weak $\ell_q$ space ($\ell_q^w = \ell_{q,\infty}$)is very close to $\ell^q$, there holds $\ell^q \subset \ell_q^w \subset \ell^{q'}$ for any $q < q'$. In particular, if $H = \ell^2(\mathcal{J}_\star)$ it holds that the weak $\ell^q$ space is equal to the approximation space $\mathfrak{A}^s_{\infty,H} = \ell_q^w(\mathcal{J}_\star)$ in the range $1/q = s + 1/2$, see [16, 33]. Before we proceed, we are going to introduce the concept of interpolation spaces which will be very useful for characterising Besov spaces. For further details, see e.g. [33].

**Definition 9.** Let $X$ and $Y$ be normed, linear spaces. It is assumed here that $Y$ is continuously embedded in $X$, i.e. $Y \subset X$ and $\|\cdot\|_X \leq C\|\cdot\|_Y$. For given $f \in X$ and some real number $t > 0$, the $K$-functional is defined as

$$K(f,t,X,Y) := \inf_{g\in Y} \left(\|f-g\|_X + t|g|_Y\right),$$

where $\|\cdot\|_X$ is a norm on $X$ and $|\cdot|_Y$ is a semi-norm on $Y$.

With the help of the $K$-functional, we define the real interpolation spaces $(X,Y)_{\theta,q}$ for $0 < \theta < 1$ and $0 < q \leq \infty$ as the set of all functions for which

$$|f|_{(X,Y)_{\theta,q}} := \begin{cases} \left( \int_0^\infty \left[ t^{-\theta} K(f,t,X,Y) \right]^q \frac{\mathrm{d}t}{t} \right)^{1/q}, & \text{if } 0 < q < \infty, \\ \sup_{t>0} t^{-\theta} K(f,t,X,Y), & \text{if } q = \infty, \end{cases} \tag{3.7}$$

is finite. Often, instead of (3.7), the following equivalent formulation is used

$$|f|_{(X,Y)_{\theta,q}} \sim \begin{cases} \left( \sum_{i=0}^\infty \left[ 2^{i\,\theta} K(f,2^i,X,Y) \right]^q \right)^{1/q}, & \text{if } 0 < q < \infty, \\ \sup_{i\geq 0} 2^{i\,\theta} K(f,2^i,X,Y), & \text{if } q = \infty. \end{cases}$$

The aforementioned approximation spaces $\mathfrak{A}^s_{q,H}$ are such interpolation spaces, i.e. $\mathfrak{A}^s_{q,H} = (H, \mathfrak{A}^r_{q',H})_{s/r,q}$ for $r > s > 0$, $0 \leq q' \leq \infty$ and $0 < q \leq \infty$, see [33, Theorem 2], if (2.6) and (2.7) hold. These interpolation spaces will prove beneficial for the upcoming subsection, where we want to state results for the interpolation between different function spaces.

### 3.2.4 Besov Spaces

As mentioned before, there are certain situations where the solution is not smooth enough any more to be contained in a Sobolev space of smoothness $s$, but is still contained in the Besov space with the same smoothness index, as Besov regularity is milder and considerably

enlarges the set of functions contained in the space. There are different classical approaches to classify regularity or smoothness. In the Hölder spaces $\mathcal{C}^s$, smoothness is measured by differences. Sobolev spaces $W_p^s$ are well known for measuring smoothness in the spaces $L^p$ by checking if the weak derivatives up to some positive integer order $s$ of a function are still contained in the space $L^p$. Sobolev spaces $W_p^s$ for non-integer $s$ in addition measure Hölder smoothness $\lfloor s \rfloor + \alpha$ with $\alpha \in [0, 1)$. For $p = 2$ the Sobolev spaces $W_p^s$ are the same as the spaces $H^s$. Sometimes these parameters do not suffice to measure smoothness. Besov spaces $B_{p,q}^s$ feature three parameters, where $s$ is the order of smoothness and $p$ gives the space $L^p$ in which the smoothness is measured. The third parameter $q$ is used to obtain a finer graduation, while leaving the parameters $s$ and $p$ unchanged. Here, we mainly introduce those aspects of Besov spaces which are important for our implementation. For a more detailed discussion on the subject, the reader is referred to, e.g. [33].

First, we introduce the characterisation of Besov spaces trough moduli of smoothness. To that end, let $\boldsymbol{h} \in \mathbb{R}^d$, $f \colon \mathbb{R}^d \to \mathbb{R}$ be a given function and $\Omega \subset \mathbb{R}^d$ a Lipschitz domain. We define the difference operator $\Delta_{\boldsymbol{h}} f \coloneqq f(\cdot + \boldsymbol{h}) - f(\cdot)$ and, for an integer $1 < r \in \mathbb{N}$, the $r$-th difference operator given by $\Delta_{\boldsymbol{h}}^r f \coloneqq \Delta_{\boldsymbol{h}}(\Delta_{\boldsymbol{h}}^{r-1} f)$. It is a convention to set the $r$-th difference operator $\Delta_{\boldsymbol{h}}^r$ to zero if any of the points $\boldsymbol{x} + k\boldsymbol{h}$ for $k = 0, \dots, r$ is not contained in $\Omega$. For a function $f \in L^p = L^p(\Omega)$ with $0 < p \leq \infty$, we define the modulus of smoothness or order $r$

$$\omega_r(f, t)_p \coloneqq \sup_{\|\boldsymbol{h}\| \leq t} \|\Delta_{\boldsymbol{h}}^r f\|_{L^p}$$

for every $t > 0$.

Functions in $L^p$ which show a similar behaviour in terms of moduli of smoothness, i.e.

$$\sup_{t>0} t^{-s} \omega_r(f, t)_p < \infty \quad \text{and } r > s,$$

are now collected in one set. For $0 < s < r$ and $0 < p, q < \infty$, the Besov space $B_{p,q}^s = B_{p,q}^s(\Omega)$ is defined as the space of all functions $f$ that fulfil:

$$|f|_{B_{p,q}^s} \coloneqq \begin{cases} \left( \displaystyle\int_0^\infty [t^{-s} \omega_r(f, t)_p]^q \frac{\mathrm{d}t}{t} \right)^{1/q} < \infty, & 0 < q < \infty, \\ \displaystyle\sup_{t>0} t^{-s} \omega_r(f, t)_p < \infty, & q = \infty. \end{cases} \tag{3.8}$$

This is a seminorm for $B_{p,q}^s$. Including $\|f\|_{L^p}$ to this seminorm gives us a quasi-norm on the Besov space. For characterising Besov spaces we can consider also the decay of coefficients of a function which is decomposed into a series of building blocks. Our interest here lies in the wavelet expansion of a function. All functions for which their coefficients in the wavelet expansion exhibit a similar decay behaviour are collected. We shall resume some important properties of Besov spaces $B_{p,q}^s$, Sobolev spaces $W_p^s = W_p^s(\Omega)$ and spaces which lie in between, see e.g. [33, 67]. To this end, we assume that the bounded domain $\Omega$ is Lipschitz.

**Lemma 10.** (i) For $p \neq 2$ and $s \notin \mathbb{N}$, the Sobolev spaces $W_p^s$ are equal to the Besov spaces $B_{p,p}^s$. If $p = 2$, the equality holds for all $s > 0$.

(ii) For $0 < p \leq p' \leq \infty$, there holds that $B_{p',q}^s \hookrightarrow B_{p,q}^s$.

(iii) For $0 < q < q' \leq \infty$, we have the embedding $B_{p,q}^s \hookrightarrow B_{p,q'}^s$, but for any $\delta > 0$ we have the embedding $B_{p,q'}^{s+\delta} \hookrightarrow B_{p,q}^s$.

(iv) For $1 \leq p, p' \leq \infty$ and $s, s' \geq 0$, we have the embedding $B_{p,p}^s \hookrightarrow B_{p',p'}^{s'}$ if the relation $s - s' \geq d\left(\frac{1}{p} - \frac{1}{p'}\right) > 0$ holds.

(v) Interpolation between $L^p$ and Sobolev spaces $W_p^s$: For $0 < \theta < 1$ and $0 < q \leq \infty$, there holds

$$(L^p, W_p^s)_{\theta,q} = B_{p,q}^{\theta s}. \tag{3.9}$$

(vi) Interpolation between Besov spaces: For $s' < s''$ and $0 < q, q' \leq \infty$, there holds

$$(B_{p,q}^{s'}, B_{p,q'}^{s''})_{\theta,q} = B_{p,q}^{\theta s},$$

for $s = (1-\theta)s' + \theta s''$ and any $0 < \theta < 1$ and $0 < q \leq \infty$.

(vii) Interpolation between Besov spaces and $L^p$: For $0 < \theta < 1$ and $0 < q \leq \infty$, there holds

$$(L^p, B_{p,q'}^s)_{\theta,q} = B_{p,q}^{\theta s}, \tag{3.10}$$

for any $0 < q' \leq \infty$.

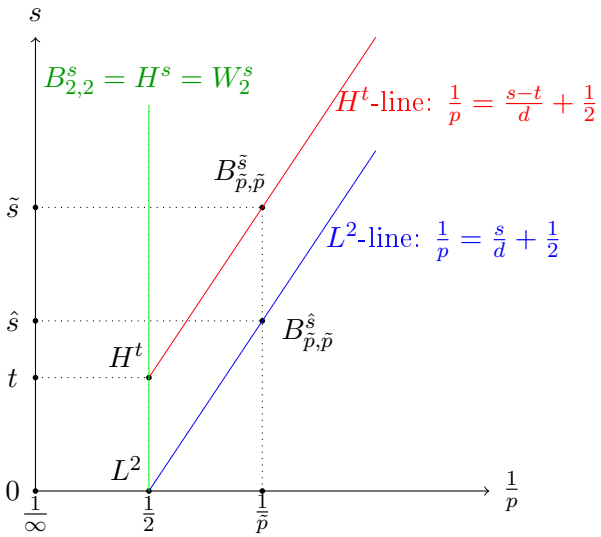



Figure 3.1: Triebel/DeVore diagram of function spaces.

The relations between different spaces from Lemma 10 can be graphically interpreted with the help of the diagram found in Figure 3.1. It has to be interpreted as follows. Each point $(1/p, s)$ in the diagram belongs to the space $B^s_{p,q}$, where the third parameter $q$ is not taken into account, i.e. each point in the diagram represents a range of Besov spaces. However, one has to be careful, as some results do not necessarily hold for any value of $q$. Whenever this is the case, we will write down the conditions for $q$. In equation (3.10), we have a look at the vertical line going through $(1/p, 0)$, for a fix integrability parameter $p$, where the space $L^p$ is represented. All spaces on this line correspond to Besov spaces with a smoothness index $0 < s' < s$ and integrability $p$. Equation (3.9) states a similar relation, but with a Sobolev space instead of a Besov space. From Lemma 10 (ii), we know that Besov spaces $B^s_{p,q}$ with larger integrability index $p$ are contained within a Besov space $B^s_{p',q}$ with $p' < p$. In the diagram this concerns the Besov spaces located to the left of $B^s_{p',q}$. Moving horizontally to the right in this diagram for a fixed $s$ increases the value of $1/p$, thus $p$ becomes smaller and the space $B^s_{p,q}$ becomes larger. Moving up vertically, the value of the smoothness parameter $s$ increases.

For the following, we restrict ourselves to Besov spaces of the form $B^s_{\tau,\tau}$. For a fixed $p$, we consider the pair $(L^p, B^s_{\tau\tau})$. If the parameters $\tau$ and $s$ are coupled through the relation

$$\frac{1}{\tau} = \frac{s}{d} + \frac{1}{p}, \tag{3.11}$$

then there holds, see e.g. [33],

$$(L^p, B^s_{\tau\tau})_{\theta,q} = B^{s\theta}_{q,q}, \quad \text{where } \frac{1}{q} = \frac{s\theta}{d} + \frac{1}{p}.$$

Here, the integer $d$ denotes the spatial dimension. In the diagram in Figure 3.1, we find two lines with slope $d$, one going through $(1/2, 0)$ and one going through $(1/2, t)$. Any Besov space which lies above the according line can be embedded into the underlying space $L^p$. Besov spaces $B^s_{\tau,\tau}$ which lie on the line can be embedded into $L^p$ if the relation (3.11) holds. The following theorem by Dahlke and DeVore [23] is one of the main results of this chapter:

**Theorem 11.** ([23, Theorem 3.2]) Let $\Omega$ be a bounded domain in $\mathbb{R}^{d+1}$ with Lipschitz boundary $\Gamma$. If $v$ is a harmonic function on $\Omega$ which is in the Besov space $B^\lambda_{p,p}$ for some $0 < p \leq \infty$ and some $\lambda > 0$, then

$$v \in B^\alpha_{\tau,\tau} \quad \text{for} \quad \tau = \left(\frac{\alpha}{d+1} + \frac{1}{p}\right)^{-1} \quad \text{and} \quad 0 < \alpha < \frac{\lambda(d+1)}{d}.$$

We recall the approximation spaces, which were introduced in Subsection 3.2.2. Suppose that the wavelets have sufficiently many vanishing moments $\tilde{m}$ and enough smoothness $\gamma$ required for the validity of the Jackson (2.6) and Bernstein estimates (2.7). Then, there holds the following characterisation for the approximation spaces $\mathfrak{A}^s_{q,H} = \mathfrak{A}^s_{q,p}$ in $H = L^p$.

Let $1 < p < \infty$, $0 < s'$ and $1/\tau = s'/d + 1/p$. For each $0 < s < s' \leq \infty$ and $0 < q \leq \infty$ holds

$$(L^p, B^{s'}_{\tau,\tau})_{s/s',q} = \mathfrak{A}^{s/d}_{q,p}.$$

For each $s$, there is a value of $q$, given by $1/q = s/d + 1/p$, for which this interpolation space is a Besov space

$$(L^p, B^{s'}_{\tau,\tau})_{s/s',q} = B^s_{q,q},$$

with equivalent quasi-norm, see e.g. [33]. When we have $H = H^s$, functions from the approximation spaces $\mathfrak{A}^{s'}_{\infty,s}$ can be characterised by Besov spaces $B^{s+ds'}_{p,p}$. We will in the following omit the first index in the subscript and write $\mathfrak{A}^{s'}_s := \mathfrak{A}^{s'}_{\infty,s}$.

We are going to conclude this subsection with two error estimates, the first of which is the well known estimate for uniform methods. Let $\mathbf{\Psi}$ be a $H^s(\Gamma)$-normalized wavelet basis and $\boldsymbol{\rho} \in \ell^2$ and recall that $(m, \tilde{m}) = (1, 3)$ for our wavelet basis.

**Lemma 12.** For $\rho := \mathbf{\Psi}\boldsymbol{\rho} \in H^{s+ds'}(\Gamma)$ with $s' \leq \bar{s} := \frac{m-s}{d}$, uniform refinement with with respect to the spaces $V_i$ with $N$ degrees of freedom yields the estimate

$$\inf_{\boldsymbol{\rho}_N \in V_i} \|\boldsymbol{\rho} - \boldsymbol{\rho}_N\|_{\ell^2(\mathcal{J}_\star)} \lesssim h^{ds'} \|\rho\|_{H^{s+ds'}(\Gamma)} \sim N^{-s'} \|\rho\|_{H^{s+ds'}(\Gamma)},$$

with $h \sim N^{-d}$.

As mentioned earlier, if the boundary $\Gamma$ has edges or the right-hand side admits singularities, the solution might not be in $H^{s+ds'}(\Gamma)$ any more. We have the following error estimate for adaptive methods.

**Lemma 13.** For $\rho \in B^{s+ds'}_{p,p}(\Gamma)$ and $s' \leq \bar{s} := \frac{m-s}{d}$ with $\frac{1}{p} = s' + \frac{1}{2}$, then adaptive refinement gives the error estimate

$$\inf_{\boldsymbol{\rho}_\mathcal{J} \in X_N} \|\boldsymbol{\rho} - \boldsymbol{\rho}_\mathcal{J}\|_{\ell^2(\mathcal{J}_\star)} \lesssim h^{ds'} \|\rho\|_{B^{s+ds'}_{p,p}(\Gamma)} \sim N^{-s'} \|\rho\|_{B^{s+ds'}_{p,p}(\Gamma)},$$

again with $h \sim N^{-d}$.

With these two results, we can conclude the following: By using nonlinear approximation, we can achieve the same convergence rate as we would by using linear approximation. However, the condition that the solution be in the Sobolev space $H^{s+ds'}$ is much more restrictive than the condition for the solution to be in the Besov space $B^{s+ds'}_{p,p}$. In regard to the diagram in Figure 3.1 this means that, instead of vertically going up the $H^s$-line (which is the case for linear approximation), we can go up the line with slope $d$. We thus work in the larger Besov space, which can still continuously be embedded into the underlying Sobolev space. Note, that the maximal rate of convergence for our piecewise constant wavelets is given by $\bar{s} < (1 - s)/d = (1 - s)/2$. As for the single layer operator $s = -1/2$ and for the double layer operator $s = 0$, this results in a maximal rate of $N^{-0.75}$ and $N^{-0.5}$, respectively.

With these fundamentals on approximation spaces in mind, we now turn towards the theoretical aspects of the implementation.

## 3.3 On Adaptive Wavelet Algorithms

The goal of our adaptive implementation is the following: For $\boldsymbol{\rho} \in \mathfrak{A}_q^s$ for some $s \leq \bar{s}$ and a target accuracy $\epsilon$, we want to compute the solution $\boldsymbol{\rho}_{\mathcal{J}}$ with $\#\mathcal{J} \lesssim \epsilon^{-1/s} \, |\boldsymbol{\rho}|_{\mathfrak{A}_q^s}$, such that $\|\boldsymbol{\rho} - \boldsymbol{\rho}_{\mathcal{J}}\| \leq \epsilon$. We assume that we have a $H^s$-normalised wavelet basis, with $s$ denoting the smoothness index of the energy space. This shall be done in optimal complexity which stays proportional to the size of the index set, i.e. $\mathcal{O}(\#\mathcal{J})$.

In this subsection, we specify what building blocks will be needed to arrive at an adaptive algorithm with optimal complexity. The adaptive algorithm with its building blocks Solve, Coarse, Apply or Rhs, has already been introduced in, e.g. [16,17,27,28,44,47,54]. The particular algorithm used for this implementation is according to Gantumur, Harbrecht and Stevenson [46]. It consists of the following steps:

$$\boxed{\textsc{Solve}} \longrightarrow \boxed{\textsc{Estimate}} \longrightarrow \boxed{\textsc{Mark}} \longrightarrow \boxed{\textsc{Refine}}$$

For an initial finite index set $\mathcal{J} \subset \mathcal{J}_\star$, the system of linear equations (3.3) is assembled and solved with accuracy $\epsilon$, i.e. $\boldsymbol{\rho}_{\mathcal{J}} = \textsc{Solve}[\mathcal{J}, \epsilon]$ where the output $\boldsymbol{\rho}_{\mathcal{J}}$ satisfies $\|\boldsymbol{\rho} - \boldsymbol{\rho}_{\mathcal{J}}\| \leq \epsilon$. Subsequently, the following routine stated in Algorithm 1, we will call it Grow, is initiated. Its goal is to estimate the residuum with sufficient accuracy $\epsilon$ by extending the incoming index set $\mathcal{J}$ to $\mathcal{J}'$ in such a way that the error is reduced by a constant factor in each iteration. After this loop, the algorithm assembles and solves the system of linear equations with respect to the larger index set $\mathcal{J} \subset \mathcal{J}''$ with accuracy $\epsilon$.

**Algorithm 1:** Grow.

Given: $\eta_{\text{init}}$, $\mathcal{J}$;
Set: $\eta = \eta_{\text{init}}$;
**do**
  Set: $\eta = \eta/2$;
  Calculate $\boldsymbol{r}_{\mathcal{J}'} = \textsc{Rhs}[\eta] - \textsc{Apply}[\eta, \boldsymbol{A}, \boldsymbol{\rho}_{\mathcal{J}}]$;
  Set: $\mathfrak{r} = \|\boldsymbol{r}_{\mathcal{J}'}\|$;
  Calculate $\boldsymbol{r}_{\mathcal{J}''} = \textsc{Coarse}[\boldsymbol{r}_{\mathcal{J}'}, \theta]$ ($0 < \theta < 1$ threshold constant);
**while** $\eta \geq \mathfrak{r}$;

Some more explanations with respect to the inner part of Grow are in order. The second step inside the do-while loop estimates the residuum up to the accuracy $\eta/2$ by computing $\boldsymbol{r}_{\mathcal{J}} = \boldsymbol{f}_{\mathcal{J}} - \boldsymbol{A}\boldsymbol{\rho}_{\mathcal{J}}$. Thereby, we have hidden a routine which is run in the process, the so-called Prediction routine, which enlarges the index set $\mathcal{J}$ to $\mathcal{J}'$. Therein, the routine $\textsc{Rhs}[\eta]$ calculates the right-hand side up to the accuracy $\eta$, i.e. $\|\boldsymbol{f} - \boldsymbol{f}_{\mathcal{J}'}\| \leq \eta$. Furthermore,

APPLY$[\eta, \boldsymbol{A}, \boldsymbol{\rho}_{\mathcal{J}}]$ calculates an $\eta$-accurate approximation to the matrix-vector product $\boldsymbol{A}\boldsymbol{\rho}$ with respect to the new index set $\mathcal{J}'$ which is applied to the solution $\boldsymbol{\rho}_{\mathcal{J}}$, with $\mathcal{J}$ the set of wavelets emerged from the previous step. This version of the adaptive algorithm includes a coarsening step in order to control the complexity. The index set $\mathcal{J}''$ becomes smaller than $\mathcal{J}'$, i.e. $\|\boldsymbol{r}_{\mathcal{J}''}\| \leq \theta\|\boldsymbol{r}_{\mathcal{J}'}\|$ for a $0 < \theta < 1$ which is small enough. However, it still holds that $\mathcal{J} \subset \mathcal{J}'' \subset \mathcal{J}'$. This last step combines the steps mark and refine, as $\mathcal{J}''$ is a refined version of $\mathcal{J}$. Finally, the index set is redefined as $\mathcal{J} := \mathcal{J}''$ and the loop begins anew. The iteration stops if $\eta \geq \mathfrak{r}$, which happens whenever the condition $(1-\theta)\|\boldsymbol{r}_{\mathcal{J}'}\| \leq \|\boldsymbol{r}_{\mathcal{J}}\| \leq (1+\theta)\|\boldsymbol{r}_{\mathcal{J}'}\|$ is fulfilled.

In the following subsections, we will discuss each of the mentioned routines in detail. Each of them shall be of optimal complexity in order to guarantee an optimal over-all complexity of the adaptive algorithm. Rather than allowing $\mathcal{J}$ to take the form of any subset out of the set ${\mathcal{J}_\star}^{\mathcal{J}_\star}$ of all subsets, it should be taken from a strict subset $\mathcal{P} \subset {\mathcal{J}_\star}^{\mathcal{J}_\star}$. Furthermore, we want the subset $\mathcal{J}$ to form a tree. For what follows, we assume that we are always under a tree constraint, i.e. $\mathcal{P} = \mathfrak{J}$. In the following subsection, we will give a precise definition of a tree and subsequently propose our version of the tree sorting algorithm based on the algorithm which was originally proposed by Binev and DeVore in [8].

### 3.3.1 Trees

Let us specify the setting we are working with.

**Definition 14.** A *graph* is a pair $G = \{\boldsymbol{\Lambda}, E\}$ consisting of nodes $\boldsymbol{\Lambda}$ (or vertices) and edges $E \subset \{(\boldsymbol{\lambda}, \boldsymbol{\lambda}') \in \boldsymbol{\Lambda} \times \boldsymbol{\Lambda} \colon \boldsymbol{\lambda} \neq \boldsymbol{\lambda}'\}$.

The graph $G$ is *connected*, if there always exists a path between two nodes $\lambda, \lambda' \in \Lambda$. It is called *undirected* if $(\lambda, \lambda') \in E$ whenever $(\lambda', \lambda) \in E$. A *cycle* consists of a set of nodes and edges such that we can start traversing through the graph by following the edges connecting the nodes, and end at the same vertex. If no edge or node is used more than once, the cycle is called a *simple cycle*.

**Definition 15.** A graph $G$ that is connected, undirected and does not contain any simple cycles, is called a *tree*.

**Definition 16.** The topmost node of a tree is called the *root*. A node which directly descends from another node is called a *child*. The inverse of a child is called the *parent*. The generalisation of a child, i.e. a node that can be reached by repeatedly going to a child, is called a *descendant*. Reversely, a node which is reached by repeatedly going up to a parent is called an *ancestor*. The set of nodes of a tree, which share the same parent, are called *siblings*. A tree can either have a fixed number of children for each node, or the number of children can vary from node to node.

**Definition 17.** Each node of a tree has a given *level*, i.e. the number of edges to pass through, to arrive at the node starting from the root which has level 0. Given the set of

all nodes, there is a maximum level, which is referred to as the *depth* of the tree. Each node with at least one child is called an *internal node*. All nodes with no further children are called *leafs*.

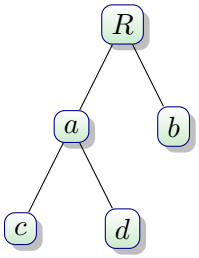



Figure 3.2: Simple example of a tree.

To illustrate these terms, consider Figure 3.2. The node $R$ is the root node of the tree. Its children are the nodes $a$ and $b$, where $b$ is a leaf of the tree and $a$ is an internal node. It has child nodes $c$ and $d$ (which are then again leafs). Conversely, $a$ is the parent of $c$ and $d$, $R$ is the parent of $a$ and $b$. Also $a$ and $b$ are siblings, as well as $c$ and $d$. Nodes $c$ and $d$ have level 2, nodes $a$ and $b$ have level 1. In this example, $c$ and $d$ are both descendants of $R$, although they are not directly connected. On the opposite, $R$ is an ancestor of $c$ and $d$. The depth of this tree is 2, since the longest distance between any node and the root node is two ($R$ to $c$ or $R$ to $d$).

After the basic terms for trees are introduced, we can proceed to a more general setting which is more appropriate for our purpose. To that end, we organise the index set $\mathcal{J}_\star$, encountered in Subsection 3.2.1, in an infinite tree. This infinite tree is supposed to have one root node and each node has a certain number of children denoted by $c_i$. The number $i \in \{0, \ldots, k\}$ is in principle allowed to vary from node to node, however, we choose $k = 3$ in our specific case. We introduce the following notation $\mathfrak{C}(\boldsymbol{\lambda})$ for the set of children of a node $\boldsymbol{\lambda}$. Similarly, with $\mathfrak{D}(\boldsymbol{\lambda})$, we denote the set of all descendants of a node $\boldsymbol{\lambda}$ and with $\mathfrak{L}(\mathcal{J})$ all leafs of the tree $\mathcal{J}$. We will use the following definition for a subtree, a proper subtree, a branch and a proper branch.

**Definition 18.** A collection of nodes with $\boldsymbol{\lambda} \in \mathcal{J}_\star$ is called a *subtree* $\mathcal{J} \subset \mathcal{J}_\star$, if for each node $\boldsymbol{\lambda} \in \mathcal{J}$ all its siblings are in $\mathcal{J}$, too. The subtree is called a *proper subtree*, if it contains all root nodes of $\mathcal{J}_\star$.

**Definition 19.** A collection of nodes $\boldsymbol{\lambda} \in \mathcal{J}_\star$ is called a *branch* $\mathcal{J} \subset \mathcal{J}_\star$, if for each node $\boldsymbol{\lambda} \in \mathcal{J}$ all its descendants are in $\mathcal{J}$ too. A *proper branch* is a branch, which has as root node any other but the root of $\mathcal{J}$.

In our implementation we have two trees, one element tree and one wavelet tree, the second of which shall be sorted by the magnitude of the wavelet coefficients. After the sorted tree is obtained, we will use a thresholding algorithm to coarse the tree. Now, we will provide

the theoretical background for efficient tree sorting, based on the article [8] of Binev and DeVore and will immediately apply it to our situation.

### 3.3.2 Coarse

The theoretical aspects of the tree-sorting, which is followed by the coarsening routine, have been considered in the article [8] by Binev and DeVore. Sorting a tree is a demanding task, which has to be done in the right way if optimal complexity shall be achieved. Recall that we have a condition for the structure of our finite subset $\mathcal{J} \subset \mathcal{J}_\star$. For our purpose, we want to preserve the tree structure of the wavelet basis. Regarding $\mathcal{J}_\star$ as an infinite tree, then $\mathcal{P}$ shall be taken from the set $\mathfrak{J}$ of all finite subtrees of $\mathcal{J}_\star$. An algorithm to find the largest $N$ coefficients, with $N$ denoting the degrees of freedom, under a tree constraint is of exponential complexity and thus prohibitive. What is done in practice, is to find a near optimal tree instead, which is nearly as good as the best $N$-term tree approximation. This procedure is referred to as near best $N$-term approximation. Finding this can be achieved by thresholding, meaning that we sort the tree by magnitude of the coefficients and keep just the largest coefficients by discarding the rest. By following the implementation according to [8], this can be done in optimal complexity $\mathcal{O}(N)$. This article discusses two different algorithms for coarsening, the second of which will be used in our implementation.

Let us introduce the error functional $e$ and the error $E$, for each node $\boldsymbol{\lambda}$ in the proper subtree $\mathcal{J} \subset \mathcal{J}_\star$

$$E(\mathcal{J}) = \sum_{\boldsymbol{\lambda} \in \mathfrak{L}(\mathcal{J})} e(\boldsymbol{\lambda}).$$

Recall that $\mathfrak{L}(\mathcal{J})$ denotes the set of all leafs of $\mathcal{J}$, $\mathfrak{C}(\boldsymbol{\lambda})$ the set of all children of a node $\boldsymbol{\lambda}$ and $\mathfrak{D}(\boldsymbol{\lambda})$ the set of all its descendants. Thereby, the error functional $e(\boldsymbol{\lambda})$ for any node $\boldsymbol{\lambda} \in \mathcal{J}$ is calculated bottom-up by

$$e(\boldsymbol{\lambda}) = |v_{\boldsymbol{\lambda}}|^2 + \sum_{\boldsymbol{\lambda}' \in \mathfrak{C}(\boldsymbol{\lambda})} e(\boldsymbol{\lambda}') = |v_{\boldsymbol{\lambda}}|^2 + \sum_{\boldsymbol{\lambda}' \in \mathfrak{D}(\boldsymbol{\lambda})} |v_{\boldsymbol{\lambda}'}|^2. \tag{3.12}$$

The routine first calculates the error functional for all leafs and subsequently, for each node, the error functional is obtained by adding up the error functional for each child. Note that cutting off a complete branch at node $\boldsymbol{\lambda}$ produces an error of size $\sqrt{e(\boldsymbol{\lambda})}$. Once the error functional is calculated for each node, we continue by calculating the modified error functional $\tilde{e}(\boldsymbol{\lambda})$ top-down. We start by setting $\tilde{e}(\boldsymbol{\lambda}) \coloneqq e(\boldsymbol{\lambda})$ for the root node. For all other nodes, we recursively set $\tilde{e}(\boldsymbol{\lambda}') \coloneqq q(\boldsymbol{\lambda})$ for all children $\boldsymbol{\lambda}' \in \mathfrak{C}(\boldsymbol{\lambda})$ with

$$q(\boldsymbol{\lambda}) \coloneqq \frac{\left[\sum_{\boldsymbol{\lambda}' \in \mathfrak{C}(\boldsymbol{\lambda})} e(\boldsymbol{\lambda}')\right] \tilde{e}(\boldsymbol{\lambda})}{e(\boldsymbol{\lambda}) + \tilde{e}(\boldsymbol{\lambda})}. \tag{3.13}$$

We notice that $\tilde{e}(\boldsymbol{\lambda}')$ is constant on all children $\boldsymbol{\lambda}' \in \mathfrak{C}(\boldsymbol{\lambda})$. More importantly, for all nodes

$\boldsymbol{\lambda}$, the coefficients $\tilde{e}(\boldsymbol{\lambda}')$ decrease monotonically for all descendants $\boldsymbol{\lambda}' \in \mathfrak{D}(\boldsymbol{\lambda})$, giving an over-all hierarchy on the wavelet tree. This is important since we do not want do destroy the tree structure by randomly deleting wavelets with small coefficients, when coarsening the tree, but rather cut off complete branches. With this modified error functional, this is automatically ensured, even if a parent itself has a small value and only one of its children has a large value.

Finding the best tree amounts to finding the minimum

$$E_N = \min_{\mathcal{J}} E(\mathcal{J}),$$

with $\mathcal{J}$ taken from the set of all proper subtrees of $\mathcal{J}_\star$ with $N$ interior nodes.

**Definition 20.** Let $\boldsymbol{v} \in \ell^2(\mathcal{J}_\star)$ be given. The *best* $\epsilon$-tree is the smallest tree $\mathcal{J} \subset \mathcal{J}_\star$ such that

$$\|\boldsymbol{v} - \boldsymbol{v}|_{\mathcal{J}}\|_{\ell^2(\mathcal{J}_\star)} \leq \epsilon.$$

As already mentioned, it is too costly to find the best $N$ terms under the tree constraint in reasonable time. What we do instead is that we want to find a proper subtree $\mathcal{J} \subset \mathcal{J}_\star$ such that $E(\mathcal{J}) \leq E_N$. $\mathcal{J}$ is then called a near optimal tree. More precisely:

**Definition 21.** Let $N$ denote the number of interior nodes of the $\epsilon$-best tree. A tree $\mathcal{J} \subset \mathcal{J}_\star$ is called *$\epsilon$-near best* if, for all $\boldsymbol{v} \in \ell^2(\mathcal{J}_\star)$, there holds

$$\#\mathcal{J} \leq CN,$$

i.e. the number of interior nodes is bounded by a constant factor times $N$ with a universal constant $C \geq 1$ for all $\boldsymbol{v} \in \ell^2(\mathcal{J}_\star)$.

The procedure, which was presented beforehand, produces a near optimal tree and, more specifically, in our case the constant $C$ which appears in Definition 21 is $C = 1$. Additionally, recall that $N_0 = 1$ in our case. With the sorted tree at hand, the next step is to coarse it. This means, we want to find the finitely supported output $\boldsymbol{v}_{\mathcal{J}} = \textsc{Coarse}\,[\epsilon, \boldsymbol{v}]$ for the finitely supported input $\boldsymbol{v}$ with tree structure, such that

$$\|\boldsymbol{v} - \boldsymbol{v}|_{\mathcal{J}}\|_{\ell^2(\mathcal{J}_\star)} \leq \epsilon,$$

for a given accuracy $\epsilon$. This shall be done within optimal complexity $\mathcal{O}(\operatorname{nnz} \boldsymbol{v})$ and is achieved by throwing away a certain percentage of the vector. Or better to say, by keeping a predefined percentage and deactivating the rest. Together with [8, Theorem 5.2] we state the following propositions for the structure of the output tree as well as the cost for computing it, cf. [8, 28].

**Proposition 22.** For any given, finitely supported input $\boldsymbol{v}$, the computational cost of the output $\boldsymbol{v}_{\mathcal{J}}$, produced by $\textsc{Coarse}[\epsilon, \boldsymbol{v}]$, remains proportional to $\#\operatorname{supp}(\boldsymbol{v})$ (where $\boldsymbol{v}$ has again tree structure) and the underlying tree $\mathcal{J}$ is near best.

The coarsening itself is realised by thresholding. To that end, first sort the input tree $\boldsymbol{v}$ by the magnitude of its modified error functional $\tilde{e}$ as explained at the beginning of the current subsection. Subsequently, these values are added up until there holds

$$\|\boldsymbol{v} - \boldsymbol{v}|_{\mathcal{J}}\|_{\ell^2(\mathcal{J}_\star)} \leq \epsilon$$

for the desired target accuracy $\epsilon$. The emerging index set $\mathcal{J}$ finally contains all relevant wavelets and the remaining wavelets are neglected. For the precise implementation of the coarsening plus the computation of the two involved functionals $e$ and $\tilde{e}$, we refer to the Subsections 4.3.2 and 4.3.10.

**Proposition 23.** If $\boldsymbol{v} \in \mathfrak{A}_q^s$ and $\|\boldsymbol{v} - \boldsymbol{w}\|_{\ell^2(\mathcal{J}_\star)} \leq \epsilon$ with $\#\mathrm{supp}(\boldsymbol{w}) < \infty$, then the output $\boldsymbol{v}_{\mathcal{J}} = \textsc{Coarse}[\epsilon, \boldsymbol{w}]$ satisfies

$$\#\ \mathrm{supp}(\boldsymbol{v}|_{\mathcal{J}}) \lesssim |\boldsymbol{v}|_{\mathfrak{A}_q^s}^{1/s} \epsilon^{-1/s},\ \|\boldsymbol{v} - \boldsymbol{v}|_{\mathcal{J}}\|_{\ell^2(\mathcal{J}_\star)} \lesssim \epsilon \quad \text{and} \quad \big|\boldsymbol{v}\big|_{\mathcal{J}}|_{\mathfrak{A}_q^s} \lesssim |\boldsymbol{v}|_{\mathfrak{A}_q^s}.$$

### 3.3.3 Apply

In this subsection, we will focus on the routine $\boldsymbol{w}_{\mathcal{J}} = \textsc{apply}\,[\epsilon, \boldsymbol{A}, \boldsymbol{v}]$, the goal of which is to compute an $\epsilon$-accurate approximation to $\boldsymbol{A}\boldsymbol{v}$. Thereby, we assume again that the support of the finitely supported input variable $\boldsymbol{v}$ is a tree. To be more precise: We wish to compute the finitely supported output $\boldsymbol{w}_{\mathcal{J}}$ for the finitely supported input $\boldsymbol{v}$ and the target accuracy $\epsilon$ such that

$$\|\boldsymbol{A}\boldsymbol{v} - \boldsymbol{w}_{\mathcal{J}}\| \lesssim \epsilon \tag{3.14}$$

within linear complexity. To that end, we first recall the vanishing moment property (2.14), which implies the so-called cancellation property of order $\tilde{m}$

$$|\langle v, \psi_{\boldsymbol{\lambda}}\rangle| \lesssim 2^{-|\boldsymbol{\lambda}|(s+\tilde{m}+1)} |v|_{W^{\tilde{m},\infty}(\mathrm{supp}(\psi_{\boldsymbol{\lambda}}))}. \tag{3.15}$$

Thereby, the term $|v|_{W^{\tilde{m},\infty}(\Omega)} \coloneqq \sup_{|\boldsymbol{\alpha}|=\tilde{m},\, \boldsymbol{x}\in\Omega} |\partial^{\boldsymbol{\alpha}} v(\boldsymbol{x})|$ denotes the semi-norm in $W^{\tilde{m},\infty}(\Omega)$. Notice, that there holds the same estimate for the dual wavelets and the number $m$ of vanishing moments of the primal wavelets.

All kernels $k(\boldsymbol{x}, \boldsymbol{y})$, which were considered in Chapter 1, are analytically standard of order $2s$, according to the following definition.

**Definition 24.** Consider a kernel $k(\boldsymbol{x}, \boldsymbol{y})$ of order $2s$, the multi-indices $\boldsymbol{\alpha} = (\alpha_1, \alpha_2)$, $\boldsymbol{\beta} = (\beta_1, \beta_2)$, and $|\boldsymbol{\alpha}| = \alpha_1 + \alpha_2$. Then the transported kernel functions $k_{i,i'}(\boldsymbol{s}, \boldsymbol{t}) \coloneqq k(\boldsymbol{\gamma}_i(\boldsymbol{s}), \boldsymbol{\gamma}_{i'}(\boldsymbol{t}))\mu_i(\boldsymbol{s})\mu_{i'}(\boldsymbol{t})$ with $1 \leq i, i' \leq M$ are called analytically standard of order $2s$ if the partial derivatives are bounded by

$$|\partial_{\boldsymbol{s}}^{\boldsymbol{\alpha}} \partial_{\boldsymbol{t}}^{\boldsymbol{\beta}} k_{i,i'}(\boldsymbol{s}, \boldsymbol{t})| \lesssim \frac{(|\boldsymbol{\alpha}| + |\boldsymbol{\beta}|)!}{(r\,\|\boldsymbol{\gamma}_i(\boldsymbol{s}) - \boldsymbol{\gamma}_{i'}(\boldsymbol{t})\|)^{(2+2s+|\boldsymbol{\alpha}|+|\boldsymbol{\beta}|)}} \tag{3.16}$$

for some $r > 0$ provided that $2 + 2s + |\boldsymbol{\alpha}| + |\boldsymbol{\beta}| > 0$.

From (3.16) and the cancellation property (3.15), we can derive the following decay estimate for the matrix entries, cf. [28,84,87,89,90],

$$|\langle \mathcal{A}\psi_{\boldsymbol{\lambda}}, \psi_{\boldsymbol{\lambda}'}\rangle| \lesssim \frac{2^{-(|\boldsymbol{\lambda}|+|\boldsymbol{\lambda}'|)(s+\tilde{m}+1)}}{\operatorname{dist}(\operatorname{supp}(\psi_{\boldsymbol{\lambda}}), \operatorname{supp}(\psi_{\boldsymbol{\lambda}'}))^{2+2s+2\tilde{m}}}. \tag{3.17}$$

By having a closer look at the above estimate, we can say that wavelets having a larger distance from each other produce a larger value in the denominator of the estimate, giving a smaller value in total. The same holds true for two wavelets having a larger sum of refinement scales $|\boldsymbol{\lambda}| + |\boldsymbol{\lambda}'|$. Note that (3.17) gives rise to the so-called first compression.

Additionally, we can state a second estimate, cf. [28,54,84,87],

$$|\langle \mathcal{A}\psi_{\boldsymbol{\lambda}}, \psi_{\boldsymbol{\lambda}'}\rangle| \lesssim \frac{2^{-|\boldsymbol{\lambda}|(s+\tilde{m}+1)}\, 2^{|\boldsymbol{\lambda}'|(1-s)}}{\operatorname{dist}(\operatorname{supp}(\psi_{\boldsymbol{\lambda}}), \operatorname{sing\,supp}(\psi_{\boldsymbol{\lambda}'}))^{2s+\tilde{m}}}, \tag{3.18}$$

where we assumed that $|\boldsymbol{\lambda}'| < |\boldsymbol{\lambda}|$, which leads to the so-called second compression. Herein, the term $\operatorname{sing\,supp}(\psi_{\boldsymbol{\lambda}})$ stands for the singular support of a wavelet (the points on which the wavelet is not smooth). In Chapter 4, we will provide some details for using these estimates for the practical realisation of the adaptive algorithm. We presented these two estimates at this point, as they are of great importance to derive a sparse representation of the infinite operator $\boldsymbol{A}$.

Let us introduce the concept of compressibility. A very important part of the implementation is the compression of the system matrix. Here compression means determining the important matrix entries and calculating these only. Being able to do this is crucial, since calculating the matrix entries is the most time consuming part in the whole implementation. In the following, by $\boldsymbol{A}_j$ we denote the compressed matrix.

**Definition 25.** The operator $\boldsymbol{A}$ is said to be $s^\star$-compressible if there exist sub-matrices $\boldsymbol{A}_j$, where the rows and columns of $\boldsymbol{A}_j$ contain at most the order of $\alpha_j 2^j$ non-zero entries, such that for any $s < s^\star$ there holds

$$\|\boldsymbol{A} - \boldsymbol{A}_j\|_{\ell^2(\mathcal{J}_\star)} \leq \alpha_j 2^{-sj}, \tag{3.19}$$

with $(\alpha_j)_{j\in\mathbb{N}}$ being a summable sequence of positive numbers with $\sum_{j\in\mathbb{N}} \alpha_j \leq 1$.

With the estimates (3.17), (3.18) at hand, we can state the following theorem [28,87].

**Theorem 26.** Suppose that the operator $\mathcal{A}\colon H^{s+\sigma}(\Gamma) \to H^{-s+\sigma}(\Gamma)$ is bounded for a sufficiently large range of positive $\sigma$. Moreover, suppose that the wavelet bases $\boldsymbol{\Psi}, \tilde{\boldsymbol{\Psi}}$ have (exactness) order $m, \tilde{m}$, respectively, and that

$$\tilde{m} > \gamma - 2s, \quad \tilde{m} > m - 2s,$$

where $\gamma$ indicates the regularity of the primal wavelets. Then, $\boldsymbol{A}$ is $s^\star$-compressible with

$$s^\star > (m-s)/2 = \bar{s}.$$

Recall that $\gamma = 1/2$ and $m = 1$ in our case. Moreover it is $s = -1/2$ for the single layer operator and $s = 0$ for the double layer operator. Hence, the highest possible convergence rate which can be achieved in the norm $\|\cdot\|_{H^s(\Gamma)}$ is $N^{-(m-s)/2}$. The computational aspects of compression itself will be discussed in the next section.

Setting $\alpha_j = (1+j)^{-2}$ in equation (3.19), gives us the estimate

$$\|\boldsymbol{A} - \boldsymbol{A}_j\|_{\ell^2(\mathcal{J}_\star)} \leq (1+j)^{-2}\, 2^{-\bar{s}j}$$

for a fixed $\bar{s} = (m-s)/2 < s^\star$. In addition, we need the following estimate for the number of non-zero entries for the $i$-th column $\boldsymbol{A}_j^i$ of $\boldsymbol{A}_j$

$$\#\mathrm{supp}(\boldsymbol{A}_j^i) \lesssim (1+j)^{-2} 2^j. \tag{3.20}$$

The precise compression pattern will be introduced in Chapter 4 about the implementation. With the compressibility at hand, we will proceed to the theory of how to compute the $\epsilon$-accurate approximation to $\boldsymbol{A}\boldsymbol{v}$. Given a finitely supported vector $\boldsymbol{v}$ with the associated index set $\mathcal{J}$, forming a tree, we can compute a sequence of nested trees $\mathcal{J}_J \subset \mathcal{J}_{J-1} \subset \cdots \subset \mathcal{J}_0 = \mathcal{J}$ such that

$$\|\boldsymbol{v} - \boldsymbol{v}|_{\mathcal{J}_j}\|_{\ell^2(\mathcal{J}_\star)} \leq 2^{j\bar{s}}\epsilon. \tag{3.21}$$

Hereby, $J$ is found by

$$J := \left\lceil \frac{\log_2\left(\|\boldsymbol{v}\|_{\ell^2(\mathcal{J}_\star)}/\epsilon\right)}{\bar{s}} \right\rceil. \tag{3.22}$$

Next, we will define the difference sets $\Delta_j := \mathcal{J}_{j-1} \setminus \mathcal{J}_j$, which will in the following be referred to as *layers*. We set the according portions of $\boldsymbol{v}$ with respect to one layer $\Delta_j$ to $\boldsymbol{v}|_{\Delta_j}$ and write $\boldsymbol{v} = \left[\boldsymbol{v}|_{\Delta_J}, \boldsymbol{v}|_{\Delta_{J-1}}, \cdots, \boldsymbol{v}|_{\Delta_1}\right]^T$. Using estimate (3.21), we deduce that

$$\|\boldsymbol{v}|_{\Delta_j}\|_{\ell^2(\mathcal{J}_\star)} = \|\boldsymbol{v}|_{\mathcal{J}_{j-1}} - \boldsymbol{v}|_{\mathcal{J}_j}\|_{\ell^2(\mathcal{J}_\star)} \leq \|\boldsymbol{v} - \boldsymbol{v}|_{\mathcal{J}_j}\|_{\ell^2(\mathcal{J}_\star)} \leq 2^{j\bar{s}}\epsilon.$$

Finally, we choose $\sum_{j=1}^{J} \boldsymbol{A}_j \boldsymbol{v}|_{\Delta_j}$ for the finitely supported approximation of $\boldsymbol{w}_\mathcal{J}$ with accuracy $\epsilon$. A schematic way of describing this matrix-vector product, is given by

$$\left[\boldsymbol{A}_J | \boldsymbol{A}_{J-1} | \cdots | \boldsymbol{A}_0\right] \begin{bmatrix} \boldsymbol{v}|_{\Delta_J} \\ \hline \boldsymbol{v}|_{\Delta_{J-1}} \\ \hline \vdots \\ \hline \boldsymbol{v}|_{\Delta_0} \end{bmatrix}$$

by setting the difference set $\Delta_0 = \mathcal{J}' \setminus \mathcal{J}$, with $\mathcal{J}'$ denoting the enlarged set obtained

by prediction. As we have chosen the layers such that estimate (3.21) holds, we have to approximate each portion $\boldsymbol{A}_j$ inside the matrix with accuracy $2^{-j\bar{s}}$ in order to fulfil the desired estimate (3.14).

We use estimate (3.19) together with the well-posedness of the underlying problem, the fact that the wavelets form a Riesz basis and the triangle inequality to derive the following estimate:

$$\begin{aligned}\|\boldsymbol{A}\boldsymbol{v}-\boldsymbol{w}_{\mathcal{J}}\|_{\ell^2(\mathcal{J}_\star)} = \left\|\boldsymbol{A}\boldsymbol{v}-\sum_{j=1}^{J}\boldsymbol{A}_j\boldsymbol{v}|_{\Delta_j}\right\|_{\ell^2(\mathcal{J}_\star)} &\leq \sum_{j=1}^{J}\|\boldsymbol{A}-\boldsymbol{A}_j\|_{\ell^2(\mathcal{J}_\star)}\|\boldsymbol{v}|_{\Delta_j}\|_{\ell^2(\mathcal{J}_\star)}\\ &\leq \sum_{j=1}^{J}\alpha_j 2^{-j\bar{s}}2^{j\bar{s}}\epsilon \leq \epsilon.\end{aligned}$$

This is the first part of the following Theorem, cf. [28].

**Theorem 27.** Let $\boldsymbol{v}$ be a finitely supported array with tree structured index set and $\epsilon > 0$. Then, the approximation $\boldsymbol{w}_{\mathcal{J}} = \text{APPLY}\,[\epsilon, \boldsymbol{A}, \boldsymbol{v}]$ is also a finitely supported tree and satisfies

$$\left\|\boldsymbol{A}\boldsymbol{v}-\sum_{j=1}^{J}\boldsymbol{A}_j\boldsymbol{v}|_{\Delta_j}\right\|_{\ell^2(\mathcal{J}_\star)} \leq \epsilon.$$

For any $s \leq \bar{s}$, one has

$$\#\text{supp}\mathcal{Z}_\epsilon \lesssim \epsilon^{-1/s}|\boldsymbol{v}|^{1/s}_{\mathfrak{A}^s_q},$$

with $\mathcal{Z}_\epsilon = \bigcup_{j=1}^{J}\bigcup_{i\in\Delta_j}\#\text{supp}(\boldsymbol{A}^i_j)$ and $\mathfrak{A}^s_q$ the approximation spaces. Furthermore, the number of arithmetic operations and storage locations is bounded by

$$W_\epsilon \lesssim \#\text{supp}(\boldsymbol{v}).$$

The estimate for the work comes from [8, Theorem 5.2]. To conclude the second estimate, we use

$$\begin{aligned}\#\mathcal{Z}_\epsilon &\leq \sum_{j=1}^{J}\sum_{i\in\Delta_j}\#\text{supp}(\boldsymbol{A}^i_j) \lesssim \sum_{j=1}^{J}\sum_{i\in\Delta_j}\alpha_j 2^j\\ &= \sum_{j=1}^{J}\#\Delta_j\alpha_j 2^j \lesssim \sum_{j=1}^{J}\alpha_j\epsilon^{-1/s}|\boldsymbol{v}|^{1/s}_{\mathfrak{A}^s_q} = \epsilon^{-1/s}|\boldsymbol{v}|^{1/s}_{\mathfrak{A}^s_q}\end{aligned}$$

using the definition of $\mathcal{Z}_\epsilon$ for the first, (3.20) for the second, and $\#\Delta_j \lesssim (2^{j\bar{s}}\epsilon)^{-1/s}|\boldsymbol{v}|^{1/s}_{\mathfrak{A}^s_q}$ cf. [28], for the third inequality.

### 3.3.4 Rhs

In order to have an approximation to the solution of the system of linear equations, we also need a routine for approximating the right-hand side $\boldsymbol{f}$ of our system. For ensuring the optimal convergence rate, the approximation to the right-hand side must converge with at least that rate. We want to have the following routine:

For the target accuracy $\epsilon$, the routine $\boldsymbol{f}_{\mathcal{J}} = \text{Rhs}\,[\epsilon]$ shall calculate the approximation $\boldsymbol{f}_{\mathcal{J}}$ to the right-hand side such that

$$\|\boldsymbol{f} - \boldsymbol{f}_{\mathcal{J}}\|_{\ell^2(\mathcal{J}_\star)} \leq \epsilon.$$

If $\boldsymbol{\rho} \in \mathfrak{A}_q^s$ for some $s < s^\star$, then it can be shown that $\boldsymbol{f} \in \mathfrak{A}_q^s$ together with $|\boldsymbol{f}|_{\mathfrak{A}_q^s} \lesssim |\boldsymbol{\rho}|_{\mathfrak{A}_q^s}$. Still, there is no way of telling how to compute such an approximation. In order to realise this in linear complexity, we need some a priori knowledge of the function $\boldsymbol{f}$. If this is given, then we have the following estimates for the work $W_\epsilon$

$$W_\epsilon \lesssim \epsilon^{-1/s} |\boldsymbol{v}|_{\mathfrak{A}_q^s}^{1/s} + 1$$

and the output $f_{\mathcal{J}}$

$$\#\text{supp}(\boldsymbol{f}_{\mathcal{J}}) \lesssim \epsilon^{-1/s} |\boldsymbol{v}|_{\mathfrak{A}_q^s}^{1/s}.$$

### 3.3.5 Solve

At the beginning of Section 3.3, we gave a quick overview of the interaction between different routines in order to form the complete adaptive algorithm. To that end, the system of linear equations is solved on the initial index set such that $\|\boldsymbol{\rho} - \boldsymbol{\rho}_{\mathcal{J}}\| \leq \epsilon$. Inside the routine grow, this initial index set $\mathcal{J}$ is enlarged to $\mathcal{J}'$ and coarsened to $\mathcal{J}''$ with $\mathcal{J} \subset \mathcal{J}'' \subset \mathcal{J}'$. This index set is sufficiently large to ensure the saturation property. Therefore, the error of the new solution $\boldsymbol{\rho}_{\mathcal{J}''}$ decreases by a constant factor compared to the error of the old solution. For calculating the new solution $\boldsymbol{\rho}_{\mathcal{J}''}$, it is sufficient to take the system matrix of the approximate matrix-vector product from the growing routine with a higher accuracy. Again, we will go into more detail for this routine in the next chapter, where the main focus lies on the implementation.

# 4

# Implementation

In Chapter 3, the main focus has been on the theoretical aspects of the adaptive wavelet method. We introduced complexity estimates and error estimates, which are needed in order to produce an adaptive algorithm of optimal complexity. The main aspect of the following chapter are the practical aspects of the implementation, for which we used the programming language C, and the computational details of our particular adaptive wavelet method.

An efficient implementation of the uniform wavelet method does already exist, see [54], and related routines Apply and Rhs can also be found therein. While the principle realisation of assembling and solving the system of linear equations is thus nothing new, the innovation of the present implementation is the conversion and fresh implementation of these routines to work with truly adaptive structures, namely trees. Though the literature cited in Chapter 3 discusses the theoretical requirements for the realisation of an adaptive wavelet scheme, we are not aware of a practical realisation by truly adaptive data structures so far.

This chapter is structured as follows: In the first section, we introduce the adaptive structures which are needed for the efficient realisation of our adaptive algorithm, incorporating the element tree and the wavelet tree. Furthermore, we will illustrate the handling of these trees, like their initialisation or the refinement and inheritance of certain characteristics from parent to child. The second section will be concerned with the practical realisation of each important routine which was mentioned in theory in Chapter 3, i.e. the realisations of Apply, Rhs, Coarse and Solve. Moreover, we focus on the implementation of further important routines like Compression, Prediction and Assembling.

## 4.1 Element Trees

In the existing uniform implementation, elements and wavelets are stored in arrays, see [54]. The advantage of this is that the access to the entries is very efficient, since we know exactly where the elements and wavelets are located in these arrays. However, this only works if we have a uniform refinement, or if there is another way of knowing the exact structure of the underlying tree. Since there is no way of knowing the exact development of such a tree in an adaptive method, we do not know in advance how many wavelets and elements (and which ones) will be needed. As we have already defined trees in the previous chapter, we will now introduce the element tree and the wavelet tree.

Before we introduce the element tree and the specific structure `Element`, used in the implementation, we will first explain what underlying mathematical structure we want to represent. To this end, we will briefly recall how to represent surfaces, as it was already mentioned in Section 2.5. For any geometry, we describe its surface $\Gamma$ as the union

$$\Gamma = \bigcup_{i=1}^{M} \Gamma_i, \quad \Gamma_i = \boldsymbol{\gamma}_i(\square), \quad i = 1, \dots, M,$$

with the diffeomorphisms $\boldsymbol{\gamma}_i$ mapping from the unit square $\square = [0,1]^2$ to the patch $\Gamma_i$. For each geometry we use, we have these mappings given in the following form:

```
typedef struct {
    Vector3     (*gamma)(Vector2 a);
    Vector3     (*nGamma)(Vector2 a);
} Parametrix;
```

The first component `(*gamma)` points to an array of functions which defines the mappings $\boldsymbol{\gamma}_i$. The second component `(*nGamma)` points to an array of functions which contains $\partial_{t_1}\boldsymbol{\gamma}_i(\boldsymbol{t}) \times \partial_{t_2}\boldsymbol{\gamma}_i(\boldsymbol{t})$ and thus encodes the surface measure $\mu_i(\boldsymbol{t}) = \|\partial_{t_1}\boldsymbol{\gamma}_i(\boldsymbol{t}) \times \partial_{t_2}\boldsymbol{\gamma}_i(\boldsymbol{t})\|$ and the normal vector $\boldsymbol{n}(\boldsymbol{\gamma}_i(\boldsymbol{t}))$. Thereby, the appearing structures `Vector2` and `Vector3` contain two and three doubles, representing a two- or three-dimensional vector, respectively.

### 4.1.1 Data Structures

In order to define the elements, we subdivide the unit square into portions of the form $\square_{j,\boldsymbol{k}} := [2^{-j}(k_1, k_1+1)] \times [2^{-j}(k_2, k_2+1)]$, where the integer $j \geq 0$ denotes the level of refinement and $\boldsymbol{k} = (k_1, k_2)$ denotes the location. We then use the mapping $\boldsymbol{\gamma}_i$ to lift it to the surface which leads to the elements $\Gamma_{i,j,\boldsymbol{k}} := \boldsymbol{\gamma}_i(\square_{j,\boldsymbol{k}})$, see Figure 2.3 in Chapter 2 for an illustration. Therefore, an element is characterised by the following properties:

```
typedef struct {
    Element     *parent, *child[4], *neighbour[4];
    int         shift_x, shift_y;
```

```
    int          level, patch, vertex[4];
} Element;
```

In each `Element`, we store its level $j$ of refinement (in `level`), the patch $i$ it belongs to (in `patch`), and its location inside the patch (in the integers `shift_x` and `shift_y`). In addition, `Element` has an array `*child[4]`, consisting of exactly four pointers indicating the children of the element, generated by one more refinement step. As long as these elements are not generated yet, thus, if the element is a leaf of the tree, we will set each pointer to `NULL`. We set also a pointer back to the `parent`, meaning the ancestor of the element in the tree. The only exception is the root node, which has its parent pointer set to `NULL`. Figure 4.1 illustrates a possible refinement of the unit square with the resulting structure of the according element tree.

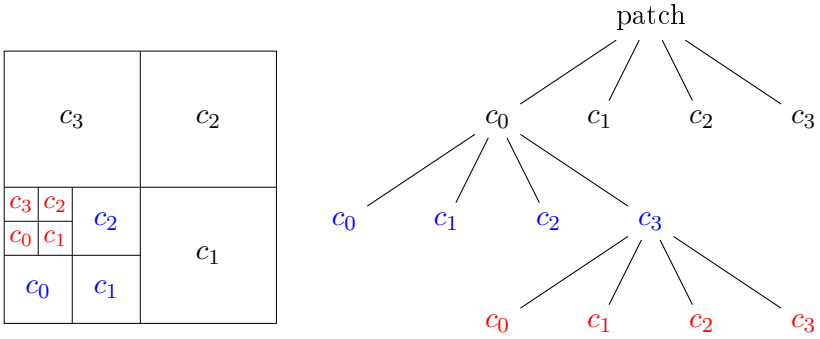



Figure 4.1: Refinement of an element with emerging element tree.

The two ingredients not discussed yet are the array `vertex[4]`, consisting of four integers which indicate where the associated vertices can be found in the array `pointList`, and the array `*neighbour[4]`, consisting of element pointers referring to the four adjacent elements of the same level. We shall next explain the initialisation of the element tree and will therein go into detail on these two arrays as well.

### 4.1.2 Initialisation of the Element Tree

As soon as we have given the parametrisations $\boldsymbol{\gamma}_i$, $i = 1, \ldots, M$, for each patch of the geometry, we can calculate the three-dimensional coordinates of the four vertices of each patch $\Gamma_i$. We do this by mapping the four corners $(0,0)$, $(1,0)$, $(1,1)$ and $(0,1)$ of the unit square to the four according vertices of each patch $\Gamma_i$ on the surface. Each point obtained in this way will then be stored in the array `pointList`, consisting of the three-dimensional coordinates of the element vertices. For each element, the array `vertex[4]` contains the four integers which indicate where the associated vertices can be found in the array `pointList`. In order to have each vertex stored in this array only once, we must not store each of the four vertices for each patch, since they all share certain edges.

We will now explain what we do in the implementation to set the array `vertex[4]` for each patch and to initialise at the same time the array `pointList`. We call the function CalcVertices for each patch, where the procedure is outlined in Algorithm 2. This function uses the associated parametrisation to calculate the (three-dimensional) image of each (two-dimensional) corner of the unit square. The function findPoint, which is called within, compares this point to all the points already stored in the array `pointList`. If the point is already contained in the array, the function returns the index where it can be found. If the point is not found, it is added to the array `pointList` and the according index is returned. Thus, after this function terminates, each patch element has the array `vertex[4]` of indices set, which contains the correct location of the associated vertex in the array `pointList`. Notice that the procedure of finding vertex indices is of quadratic cost, which is why we use it on the patch level only.

**Algorithm 2:** Calculate and set the vertices for each patch.

**for** $i = 1$ **to** `noPatches` **do**
  `e` is the $i$-th patch;
  `e` $\rightarrow$ `vertex[0]` = findPoint($\gamma_i(0,0)$, `pointList`);
  `e` $\rightarrow$ `vertex[1]` = findPoint($\gamma_i(1,0)$, `pointList`);
  `e` $\rightarrow$ `vertex[2]` = findPoint($\gamma_i(1,1)$, `pointList`);
  `e` $\rightarrow$ `vertex[3]` = findPoint($\gamma_i(0,1)$, `pointList`);
**end**

The next issue to be addressed is setting the adjacent elements for each patch, called neighbours, meaning the elements that share an edge. For setting the neighbours of a patch, we call the routine CalcNeighbours, which is outlined in Algorithm 3. For each edge of the patch, we call the routine findNeighbour. It checks whether the incoming edge, which is uniquely characterised by the two integers of its vertices in ascending order, coincides with the edge of any other patch. If they do, we have found the element of interest and set it as neighbour, if not, we set the pointer to `NULL`. Thus, the implementation can handle closed surfaces as well as non-closed ones.

**Algorithm 3:** Set the neighbours for each patch.

**for** $i = 1$ **to** `noPatches` **do**
  `e` is the $i$-th patch;
  `e` $\rightarrow$ `neighbour[0]` = findNeighbour(`e` $\rightarrow$ `vertex[0]`, `e` $\rightarrow$ `vertex[1]`);
  `e` $\rightarrow$ `neighbour[1]` = findNeighbour(`e` $\rightarrow$ `vertex[1]`, `e` $\rightarrow$ `vertex[2]`);
  `e` $\rightarrow$ `neighbour[2]` = findNeighbour(`e` $\rightarrow$ `vertex[2]`, `e` $\rightarrow$ `vertex[3]`);
  `e` $\rightarrow$ `neighbour[3]` = findNeighbour(`e` $\rightarrow$ `vertex[3]`, `e` $\rightarrow$ `vertex[0]`);
**end**

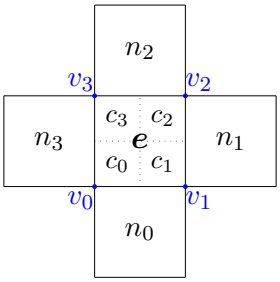



Figure 4.2: Element $e$ with neighbours $n_0$, $n_1$, $n_2$, $n_3$, children $c_0$, $c_1$, $c_2$, $c_3$ and vertices $v_0$, $v_1$, $v_2$, $v_3$.

Before we proceed to element refinement, we will explain in which way we number vertices, neighbours and children of an element $e$. At this point, we remember that we do not work directly with the three-dimensional vertices of the element, but with the integers that are associated to their location in the array `pointList`. The numbering of vertices and children is always the same, starting with the south-western vertex $v_0$ and the child $c_0$ located there, and proceeding counter-clockwise to the north-western vertex of the element. Moreover, the element that shares the edge with vertices $v_i$ and $v_{(i+1) \bmod 4}$ will always be the neighbour $n_i$. Hence, $n_0$ is the neighbour in the south, $n_1$ is the neighbour in the east, $n_2$ is the neighbour in the north and $n_3$ is the neighbour in the west. The same can be done with the vertices, using south-west, south-east, north-east and north-west instead. Figure 4.2 gives an illustration of the described situation.

After the element tree is initialised, we have at hand the list with the patch vertices on the surface, as well as the neighbouring relations among the patches. Two important remarks have to be made at this point. First, the cost of the initialisation is of order $\mathcal{O}(1)$ since the number $M$ of patches is fixed and independent of the degrees of freedom. Second, the patches are consistently oriented, but it is in general impossible to also number the neighbours consistently. We will explain what we mean with this by an example. Consider the situation illustrated in Figure 4.3, where three patches share a common vertex. Such degenerated vertices appear often when we want to parametrise a closed three-dimensional surface. Consider the element $e$ with its vertices $v_0$, $v_1$, $v_2$ and $v_3$ and its neighbours set. Element $e$ then is its western neighbour and element $e$ is its southern neighbour. The element $e$ itself is also the northern neighbour of $e$. The neighbouring relations between $e$ and $e$ are still consistently set with $e$ being the western neighbour of $e$ and $e$ being the eastern neighbour of $e$. We can thus apply the rule:

```
e → neighbour[i] → neighbour[(i+2) mod 4] = e;
```

This is not possible any more for $e$ and $e$. As we have already mentioned, $e$ is the western neighbour of $e$ but $e$ is the northern neighbour of $e$ (instead of being the eastern neighbour). Of course, this is considered in the routine FINDNEIGHBOUR, ensuring that the neighbours are set for each patch consistently in its reference.

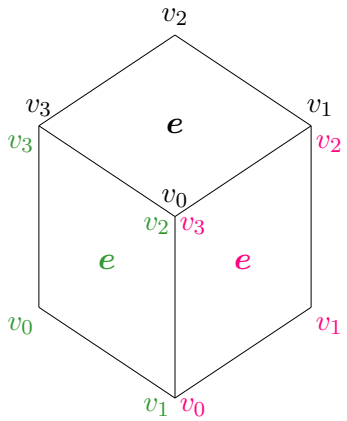



Figure 4.3: Parametrisation of a degenerated vertex.

### 4.1.3 Element Refinement

Each time we decide to refine an element, we need an efficient way to pass all the parameters required in the definition of an element to its children. Whenever an element is refined, there appear new vertices which have to be added to the array `pointList`. Thereby, we have to keep track of the vertices already available, continuously updating the array `pointList` as the code proceeds.

We will present an example to explain the element refinement. The cost of element refinement is constant in our case, meaning $\mathcal{O}(1)$, where the key ingredient for having this are the neighbourhood relations provided by the array `*neighbour[4]`. Consider the situation illustrated in Figure 4.4. We have given the element $e$ with the four known vertices $v_0$, $v_1$, $v_2$ and $v_3$, and the four neighbours $n_0$, $n_1$, $n_2$ and $n_3$. In this situation, we assume that the neighbouring element $n_2$ is already refined and rotated differently from $e$.

When the element $e$ is refined, the new child elements $c_0$, $c_1$, $c_2$ and $c_3$ will be generated, together with the possibly new vertices $p_0$, $p_1$, $p_2$ and $p_3$ on the edges and the midpoint $m$. As the vertices of the parent element $e$ are already given, we can transmit the associated vertex indices by the following rule:

```
e → child[i] → vertex[i] = e → vertex[i];
```

The midpoint $m$ is always a new vertex. Thus, we use the mapping $\gamma_i$ of the according patch to obtain the new three-dimensional point on the surface, add it to the array `pointList` and set:

```
e → child[i] → vertex[(i+2) mod 4] = pointListLength;
```

Here, `pointListLength` denotes the current length of the array `pointList`, which is increased by one after the index is set. Another relation we know for sure is that we have at least two new neighbours for each child, which we set according to the following rule:

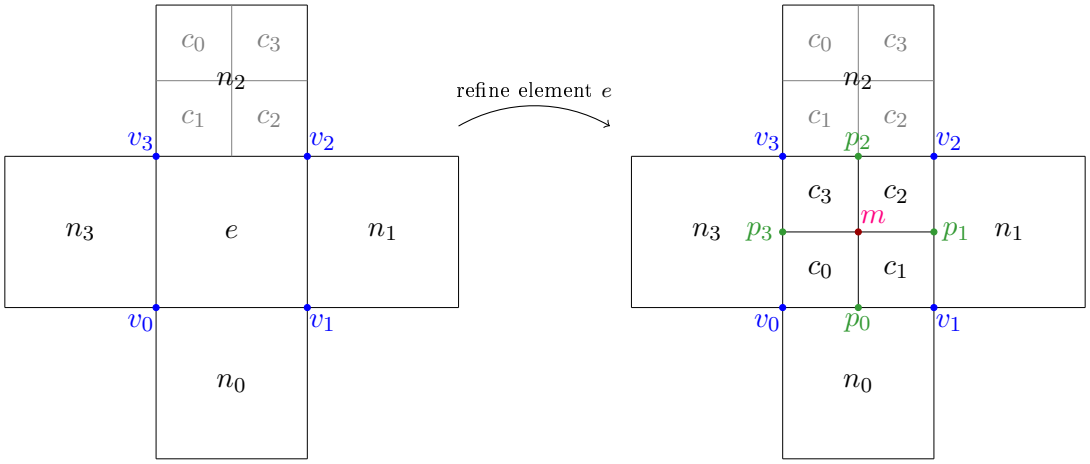



Figure 4.4: Refinement procedure of an element.

```
e → child[i] → neighbour[(i+1) mod 4] = e → child[(i+1) mod 4];
e → child[i] → neighbour[(i+3) mod 4] = e → child[(i+3) mod 4];
```

Setting the vertices $p_0$, $p_1$, $p_2$ and $p_3$ needs some more thought, since it is not clear whether these vertices already exist. This is what we need to find out without running through the whole array `pointList` each time and here is where we will exploit the neighbouring relations.

If a neighbour $n_i$ already has children, then the vertex $p_i$ already exists and we just have to find the index of this vertex. The only difficulty remaining is that the relation `e → neighbour[i] → neighbour[(i+2) mod 4] = e;` does not necessarily hold, as we have already noticed from the example in the previous section. Thus, in case the neighbour already has children, it comes down to finding out how the neighbours are related to each other. Suppose we know the index $j$ which satisfies `e → neighbour[i] → neighbour[j] = e`. Then, since the patches of the geometry are always oriented counter-clockwise, we can set the new vertex and the associated neighbouring relations according to the following rules:

```
e → child[i] → vertex[(i+1) mod 4] =
e → child[(i+1) mod 4] → vertex[i] =
e → neighbour[i] → child[j] → vertex[(j+1) mod 4];

e → child[(i+1) mod 4] → neighbour[i] = e → neighbour[i] → child[j];
e → child[i] → neighbour[i] = e → neighbour[i] → child[(j+1) mod 4];

e → neighbour[i] → child[j] → neighbour[j] = e → child[(i+1) mod 4];
e → neighbour[i] → child[(j+1) mod 4] → neighbour[j] = e → child[i];
```

If the neighbour $n_i$ does not have any children, we know that the vertex $p_i$ is new. Thus,

we use the mapping $\boldsymbol{\gamma}_i$ to obtain its three-dimensional image on the surface, add it to the list of points and set:

```
e → child[i] → vertex[(i+1) mod 4] =
e → child[(i+1) mod 4] → vertex[i] = pointListLength;
```

Recall that we allow only elements as neighbours which have the same level as the element itself. This is why some neighbours cannot and therefore must not be set at this point. In Figure 4.4, the element $c_2$ has the native neighbours $n_0 = c_1$ and $n_3 = c_3$, as well as the neighbour $n_2 = c_2$. The eastern neighbour of $c_2$ is set to `NULL` since $n_1$ is too large to be its neighbour (neighbour $n_1$ is already the neighbour of the element $e$). As we set new neighbours and points consistently, we can throughout the implementation rely on the facts described above for managing the mesh, without searching vertices in the array `pointList`.

## 4.2 Wavelet Trees

In this section, we are going to introduce the wavelet tree. To that end, let us recall the basic facts about the wavelets on surfaces from Section 2.5. We first need wavelets on the unit square, as introduced in Section 2.4. We then need mappings $\boldsymbol{\gamma}_i$ from the unit square to each patch $\Gamma_i$, $i = 1, \ldots, M$, of the surface $\Gamma$.

For wavelets $\psi^{\square}_{j,\boldsymbol{k},\iota}$ on the unit square, the integer $j$ indicates the level of refinement, $\boldsymbol{k} = (k_1, k_2)$ the location inside the unit square and $\iota$ gives the index set $\nabla_{j,\iota}$, associated with the prototype of the wavelet, out of which $\boldsymbol{k}$ is taken. Each wavelet $\psi^{\square}_{j,\boldsymbol{k},\iota}$ can be represented via a linear combination of scaling functions $\phi^{\square}_{j,\boldsymbol{k},\iota}$. In the present case of piecewise constant wavelets, the scaling functions are box functions of height $2^j$, associated with a single element of refinement level $j$. The coefficients in the linear combination are called weights. Thus, in our implementation, we represent the support of a wavelet $\psi^{\square}_{j,\boldsymbol{k},\iota}$ on the unit square by the union

$$\operatorname{supp} \psi^{\square}_{j,\boldsymbol{k},\iota} = \bigcup_{(j',\boldsymbol{k}')\in\mathcal{I}} \square_{j',\boldsymbol{k}'}, \tag{4.1}$$

with $\mathcal{I}$ being the set of elements in the wavelet's support and $\square_{j',\boldsymbol{k}'}$ denoting the elements where the wavelet is smooth. Finally, the wavelets on the surface are obtained by lifting the wavelets on the unit square to $\Gamma$ by the mapping $\boldsymbol{\gamma}_i$, which requires only the information of the index $i$.

Our data structure `Wavelet` consists thus of the following ingredients:

```
typedef struct {
    Wavelet     **child, *parent;
    Element     **support;
    int         protoType, patch;
```

```
} Wavelet;
```

In order to have a tree structure among the wavelets, each wavelet has an array of pointers, leading to the descendants of the wavelet, and a pointer back to the ancestor. The array `**child` of descendants consists of wavelets, where, in contrast to the elements (always having four children), the number of wavelet children can vary. This is why we use the pointer `**child` instead of e.g. `*child[4]`. The variable `*parent` is a pointer to the ancestor and consists of the wavelet's parent in the tree. We store the patch, on which the wavelet is located, in the variable `patch`. Subsequently, all the elements $\square_{j,\boldsymbol{k}}$ from equation (4.1), more precisely a pointer to all these elements, are being stored in the array `**support`. The details on how to compose this array exactly will be introduced in Section 4.2.2. In particular, this will be closely linked to the prototype of the wavelet, which is indicated by the variable `protoType`.

### 4.2.1 Initialisation of the Wavelet Tree

In Section 4.1, we outlined the initialisation of the element tree. Initialising the wavelet tree is performed similarly. On each patch $\Gamma_i$, $i = 1, \ldots, M$, we have given the coarse-grid scaling function $\phi^{\Gamma_i}_{0,(0,0)} = \phi^{\square}_{0,(0,0)}(\boldsymbol{\gamma}_i^{-1})$, forming the root nodes of the future wavelet tree. For each patch `p`, we set for the associated wavelet `w`:

```
w → support = p;
w → child[i] = NULL;
w → parent = NULL;
```

One more ingredient in the element tree, which has not yet been mentioned in Section 4.1, is the array `**wavelet`, containing pointers to each wavelet which has this element in its support. At the same time as the support is being set in the wavelet `w`, a pointer to this wavelet is set in the according array of each element. As soon as the initialisation of the wavelet tree is completed, each further contact with the wavelet tree comes down to the refinement of a leaf in this tree. In order to understand how wavelet refinement works with our tree, we will first introduce the so-called wavelet archive, which builds the foundation for understanding how exactly we represent our wavelets. With this knowledge at hand, we can finally continue to Section 4.2.3, where we will explain in detail how wavelet refinement works.

### 4.2.2 Wavelet Archive

The idea of the wavelet archive is to store all wavelet properties which are independent of the refinement, i.e. which are induced by translation or dilation, as global variables, such that we can access these variables whenever desired. Also, we want the implementation to be able to work independently of the different wavelet bases available.

Before we can start explaining the specific variables appearing in the archive, we have to introduce the concept of the prototype `protoType`, which is to be found in the structure `Wavelet`, mentioned earlier. For this purpose, we recall the construction of wavelet bases on the unit interval, as it has been introduced in Subsection 2.4, and the scaling functions $\phi_{j,k}^{[0,1]}$ from equation (2.8). We will focus on the wavelets with optimised support and briefly repeat the formulae for constructing them by taking tensor products of one-dimensional functions on the interval. We have given the three products

$$\begin{aligned}\phi_{j,\boldsymbol{k}}^{\square}(\boldsymbol{x}) &= \phi_{j,k_1}^{[0,1]}(x_1)\cdot\phi_{j,k_2}^{[0,1]}(x_2),\\ \psi_{j,\boldsymbol{k},1}^{\square}(\boldsymbol{x}) &= \psi_{j,k_1}^{[0,1]}(x_1)\cdot\phi_{j,k_2}^{[0,1]}(x_2),\\ \psi_{j,\boldsymbol{k},2}^{\square}(\boldsymbol{x}) &= \phi_{j+1,k_1}^{[0,1]}(x_1)\cdot\psi_{j,k_2}^{[0,1]}(x_2).\end{aligned}$$

For each of these products, we will assign a different prototype. Hereby, the prototype of the scaling functions $\phi_{j,\boldsymbol{k}}^{\square}$ is defined to be 0 and the prototype of the wavelet $\psi_{j,\boldsymbol{k},1}^{\square}$ is defined to be 1. The wavelets $\psi_{j,\boldsymbol{k},2}^{\square}$ are somewhat special. Note that we have taken the scaling functions $\phi_{j+1,k_1}^{[0,1]}$ on level $j+1$ for their construction, ending up with twice as many wavelet functions, which have, however, half the support compared to the wavelets of prototype 1. We will thus assign the prototypes 2 and 3 for these wavelets. Later in this section, we will see why this choice is especially convenient for our implementation.

Now, let us introduce the first two arrays used in the wavelet archive:

```
extern int*     supportSize[];
extern double** waveWeights[];
```

The array `supportSize` contains integers that indicate the number of elements in a wavelet's support. The support size of a wavelet depends on its prototype. Therefore, the overall size of this array corresponds to the number of different prototypes. The variable `waveWeights` contains the values of the weights of each element contained in the wavelet's support. The size of this array thus depends on the number of elements in a wavelet's support and its prototype, resulting in a two-dimensional array. Figure 4.6 shows the weights for each prototype in case of the two-dimensional Haar wavelets. Here and in the following, we use the top view onto a scalar function in $\mathbb{R}^2$ to present our wavelets, as seen in Figure 4.5.

As shown in Figure 4.6, each of the prototypes has a possibly different set of weights and number of supporting elements. Since these properties alone are not sufficient to uniquely determine a wavelet, we introduce the next variable in the archive:

```
extern char**   waveSupport[];
```

This array consists of a set of characters, indicating the elements in the wavelet's support with respect to one well-defined reference element, which we will, in the following, call the master element. In order to fully understand this, we need to introduce another wavelet property not mentioned yet, namely the pointer

```
Element *master;
```

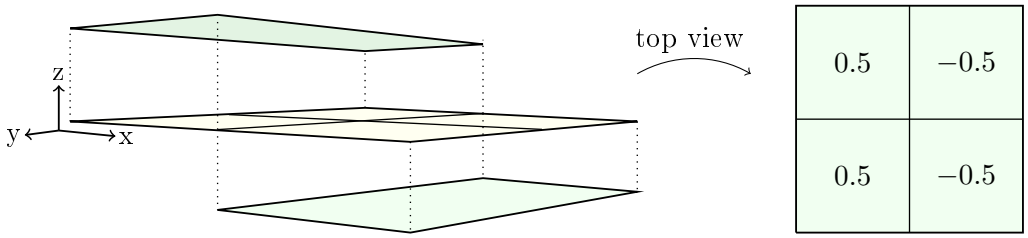



Figure 4.5: Three-dimensional graph of a Haar wavelet and how it is drawn in top view.

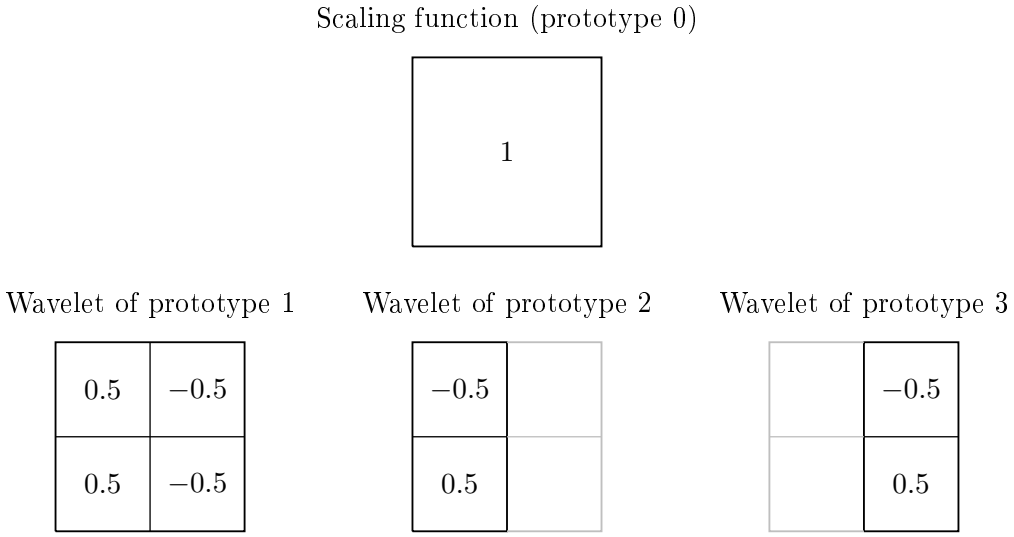



Figure 4.6: Top view of the prototypes in case of Haar wavelets.

which contains the pointer to exactly this master element. Thus, for each wavelet, we assign a master element in addition to its supporting elements, with respect to which we can set the support, only needing the knowledge on its prototype. In order to illustrate this, let us have another look at Figure 4.6. For each of the four prototypes, we set the pointer `Element *master` to the coarse element which coincides with prototype 0. The wavelet with prototype 1 then has this element's four children as support, the wavelet of prototype 2 is supported by the element's children $c_0$ and $c_3$, and the one with prototype 3 is supported by its children $c_1$ and $c_2$. For the two-dimensional Haar basis, this leads to the following specific arrays:

```
int* supportSize[4] = {
    (int[1]) {1},
    (int[1]) {4},
    (int[1]) {2},
    (int[1]) {2}
};
```

```
double** waveWeights[1] = { (double*[4]) {
    (double[1]) { 1.0 },
    (double[4]) { 0.5, -0.5, -0.5, 0.5 },
    (double[2]) { 0.5, -0.5 },
    (double[2]) { 0.5, -0.5 } }
};

char** waveSupport[4] = {
    (char*[1]) { "e" },
    (char*[4]) { "c0", "c1", "c2", "c3" },
    (char*[2]) { "c0", "c3" },
    (char*[2]) { "c1", "c2" }
};
```

Each of the four rows in the arrays corresponds to one wavelet prototype. Aside from the level dependence of a wavelet, all wavelets of the same prototype have the same structure, i.e. the same support with respect to their particular master element and the same weights. The only remaining difference is that wavelets on a larger refinement scale have smaller supporting elements, resulting in a smaller support in total. For our purpose, it is useful to represent a wavelet $\psi_{\boldsymbol{\lambda}}$ independently of its level $j$. The piecewise constant part $\phi^{\square}_{j',\boldsymbol{k}'}(\boldsymbol{s})$ on each element $\square_{j',\boldsymbol{k}'}$ in the support is multiplied with the (level independent, but prototype dependent) weight $\omega_{\boldsymbol{k}',\iota}$:

$$\psi_{\boldsymbol{\lambda}}(\boldsymbol{\gamma}_i(\boldsymbol{s})) = \sum_{(j',\boldsymbol{k}')\in\mathcal{I}} \phi^{\square}_{j',\boldsymbol{k}'}(\boldsymbol{s})\,\omega_{\boldsymbol{k}',\iota}. \tag{4.2}$$

As the level $j$ is already stored with the wavelet, we have to store only one set of weights $\omega_{\boldsymbol{k}',\iota}$ for each prototype in a global variable. As these weights are independent of the level $j$, this requires only $\mathcal{O}(1)$ memory. Furthermore, by knowing the wavelets' prototype, their weights can be accessed easily by the global variable.

### 4.2.3 Wavelet Refinement and Inheritance

With the arrays from the archive, the support and the weights of each wavelet are accessible at any time. Whereas we considered one wavelet in isolation in the aforementioned situation, we will now examine how the variables `**support`, `protoType` and `*master` are transferred to a wavelet's children.

Since we had not introduced the variables `protoType` and `*master`, when we first talked about the initialisation of the wavelet tree, we have to jump back and explain how to do this when initialising the tree. Recall that the pointer `*master` is set to the patch for each scaling function and its prototype is set to 0. In particular, the scaling function is the only wavelet, which has the pointers `**support` and `*master` set to the same element. It is also

the only wavelet of prototype 0.

As soon as we have set all these variables for the scaling functions, we need one more ingredient in the archive to be able to refine wavelets and set all desired variables of their children:

```
extern int**        inherit[];
```

This three-dimensional array contains two different kinds of inheritance information for all available wavelet prototypes, where each line of the array corresponds to a different prototype. Thus, each prototype has its own set of inheritance rules for wavelet refinement. The first 4-tuple of integers assigns the prototype of a wavelet's children. The second 4-tuple of integers is used to set the pointer `*master` for each of the four children.

Since the array `**inherit[]` may seem somewhat unintuitive at first, we will use an example to explain its handling and usefulness in detail. Let us therefore consider the Haar basis, in which case the array for inheritance is the following:

```
int** inherit[4] = {
    (int*[2]) { (int[3]) {1,2,3},   (int[3]) {0,0,0} },
    (int*[2]) { (int[4]) {1,1,1,1}, (int[4]) {0,1,2,3} },
    (int*[2]) { (int[4]) {2,3,3,2}, (int[4]) {0,0,1,1} },
    (int*[2]) { (int[4]) {2,3,3,2}, (int[4]) {0,0,1,1} }
};
```

We will go into details shortly, but first we illustrate how this array is used in a refinement step. Suppose that we have given a wavelet of prototype 2. This means that we are looking at the third line

```
    (int*[2]) { (int[4]) {2,3,3,2}, (int[4]) {0,0,1,1} }
```

of the array `**inherit[]`.

The first 4-tuple, which is `{2,3,3,2}`, encodes the `protoType` of the four children of the wavelet with prototype 2. They are numbered counter-clockwise like the children of an element. Hence, for this set of integers, the wavelet children 0 and 3 receive the prototype 2, while the wavelet children 1 and 2 receive the prototype 3.

The second 4-tuple, which is `{0,0,1,1}`, will help to identify the pointer `*master` of each child by the following rule:

```
w → child[i] → master = w → support[inherit[w → protoType][1][i]];
```

We notice that the use of this 4-tuple of integers is not as straightforward as for the first 4-tuple of integers, as we insert the according integer into the support of the parent wavelet. In this particular example (with the array `{0,0,1,1}`), this means that both, child 0 and child 1, receive the parent's first supporting element as pointer `*master`. Child 2 and child 3 receive the parent's second supporting element.

Recall Section 4.2.2, where we anticipated that we will assign both, prototypes 2 and 3, to the wavelets $\psi^{\square}_{j,\boldsymbol{k},2}$. Figure 4.6 illustrates the situation. The wavelet which is located at the left (plot in the middle) is of prototype 2, whereas the wavelet at the right (right plot) is of prototype 3. The reader may ask why these two wavelets, which look much alike, have two different prototypes. If we would choose just prototype 2 for both wavelets, we would run into consistency problems when transferring some properties to the wavelet's children, the support for instance, found in `waveSupport`. Assigning two different prototypes allows us to identify each wavelet uniquely and, as a consequence, transfer all properties to its children.

We notice that the first line of the array `**inherit[4]` contains two triples of integers instead of two 4-tuples, which is because the first line corresponds to wavelets of prototype 0. This is a unique property of the scaling function: It is the only function of prototype 0, which has three children, one of each prototype 1, 2 and 3.

To summarise the current subsection, we give an outline of the wavelet refinement in Algorithm 4. The function SetSupp, called in the routine, sets the support for each child by using the global variables `supportSize` and `waveSupport`.

**Algorithm 4:** Wavelet refinement.

```
for i = 0 to noChildren-1 do
    w → child[i] → parent = w;
    w → child[i] → protoType = inherit[w → protoType][0][i];
    w → child[i] → master = w → support[inherit[w → protoType][1][i]];
    setSupp(w → child[i], supportSize[w → child[i] → protoType]);
end
```

### 4.2.4 Wavelets with Three Vanishing Moments

For the sake of simplicity, we deliberately used the Haar basis to introduce the archive. The situation changes considerably when we decide to use wavelets with more than one vanishing moment. For instance, the interior and boundary wavelets are the same for Haar wavelets. This is no longer the case for wavelets with three vanishing moments. This subsection will address these challenges by discussing how to design the archive in a way which is independent of the chosen wavelet basis.

First, recall Figure 2.2, showing the situation on the interval. We see the interior wavelet with three vanishing moments in the middle, as well as the modified left and right boundary wavelet, respectively. By using the same strategy as for the Haar basis in Subsection 2.4.2 in order to construct wavelets with three vanishing moments on the unit square, we obtain three prototypes for the interior wavelets (type zero is again assigned only to the scaling function). Additionally, we get two-dimensional boundary wavelets at each boundary of the unit square, each of them defining a new prototype. This leads to nine prototypes in

total, all of which are displayed in Figure 4.7 together with the according weights. The master element is coloured slightly darker than the other elements in the support.

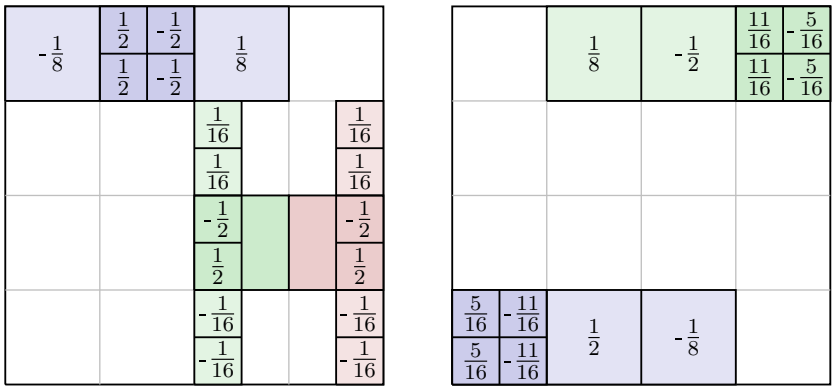
The elements, which are in the support of a wavelet, are given in the array `waveSupport`, where each line corresponds to a different prototype, defined with respect to the master element:

```
char** waveSupport[10] = {
    (char*[1]) { "e" },
    (char*[6]) { "c0" , "c1" ,  "c2"  ,  "c3"  ,  "n3"  ,  "n1"  },
    (char*[6]) { "c0" , "c3" , "n0c0" , "n0c3" , "n2c0" , "n2c3" },
    (char*[6]) { "c1" , "c2" , "n0c1" , "n0c2" , "n2c1" , "n2c2" },
    (char*[6]) { "c0" , "c1" ,  "c2"  ,  "c3"  ,  "n1"  , "n1n1" },
    (char*[6]) { "c0" , "c1" ,  "c2"  ,  "c3"  ,  "n3"  , "n3n3" },
    (char*[6]) { "c0" , "c3" , "n2c0" , "n2c3" ,"n2n2c0","n2n2c3"},
    (char*[6]) { "c1" , "c2" , "n2c1" , "n2c2" ,"n2n2c1","n2n2c2"},
    (char*[6]) { "c0" , "c3" , "n0c3" , "n0c0" ,"n0n0c3","n0n0c0"},
    (char*[6]) { "c1" , "c2" , "n0c2" , "n0c1" ,"n0n0c2","n0n0c1"}
};
```

As already mentioned in Section 2.3.3, the wavelets with three vanishing moments are supported on three elements, which is why we cannot use them before a certain level of refinement is reached. However, as soon as the necessary level of refinement is reached, the arrays for the support and the weights can be used as usual together with the same routines for transferring the prototype, the support, the weights and the master element. The only difference is that the arrays are larger, as they feature nine prototypes instead of three.

### 4.2.5 Switching Between Wavelet Bases

As already mentioned, a certain level of refinement is required to use the wavelets with three vanishing moments. Instead of using a single-scale basis up to the desired refinement scale (being 2 in this case), we will start the algorithm by using Haar wavelets until then. The use of two different wavelet bases poses a challenge, as we have to switch from Haar wavelets to the wavelets with three vanishing moments during runtime. The transfer of `support`, `protoType` and `master`, before and after changing the wavelet basis, works as explained before. The only challenge is the transfer of these variables in the moment when the basis changes. For this purpose, we will extend the archive by two new arrays and one new function. The array `kind` contains the information about when to change wavelets, the array `int* switchRules[][][]` contains a list of possible prototypes to assign to a wavelet's children during the change. Also, we will enhance the two known arrays `supportSize` and `waveSupport` by an additional parameter, such that all information for the old and the new wavelets is contained. The remaining arrays are simply enhanced with additional information on the new prototypes which appear.



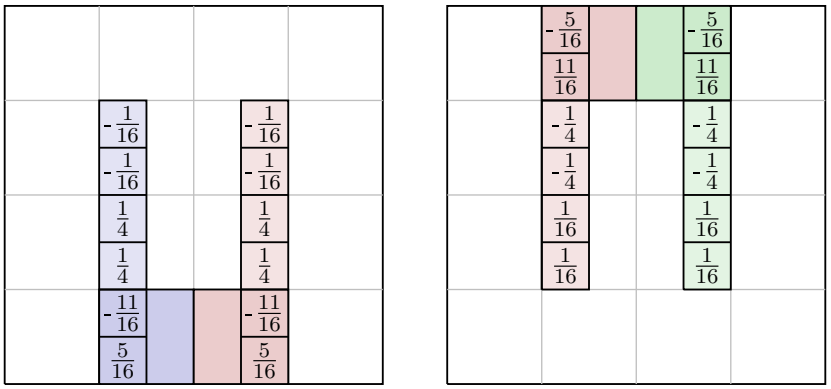




Figure 4.7: All nine prototypes of the wavelets with three vanishing moments.

We best use the particular example of changing from Haar wavelets to the wavelets with three vanishing moments to explain this situation. In this case, the array `switchRules` is defined as

```
int* switchRules[3] = {
    (int[3]) {1, 4, 5},
    (int[3]) {2, 6, 8},
    (int[3]) {3, 7, 9}
};
```

where the integers therein mean the following: Each line of the array stands for a prototype of the wavelet basis, which is used first (being Haar wavelets here). Let us consider the first line of this array which contains the numbers 1, 4 and 5. Having a Haar wavelet of type 1, these numbers give us the possible prototypes of its children after switching to the second wavelet basis. For example, a Haar wavelet of type 1 has potential children of types 1, 4 or 5, but not every prototype is allowed for every wavelet child. As soon as

the variable `kind` initialises a change between different wavelet bases, the routine `FitsIn`, outlined in Algorithm 5, is called inside the routine for wavelet refinement. It checks which of the three possible prototypes is allowed for the wavelet child under consideration. The function thereby runs through the support of each possible prototype of the wavelet's children obtained from the array `waveSupport`. The function RELATIVENEIGHBOURS, which is called within, checks if all elements in the support are located on the same patch. If they all are, then the wavelet fits in.

Figure 4.8 shows two different situations. First, we consider a wavelet `w1` of type 1 (hatched area) in the top left corner. This wavelet has four children and Figure 4.8 illustrates the situation for child 3. The possible prototype for this child are the types 1, 4 and 5. We notice that a wavelet of type 1 does not fit as a child, since part of its support would be located outside the patch. The same would happen for a child of type 5. Thus, the only possible prototype of `w1` $\rightarrow$ `child[3]` is type 4. The second wavelet `w2` under consideration is a type 3 wavelet (dotted area) and Figure 4.8 shows the situation for its child 1, which has the possible prototypes 3, 7 and 9. Again, for a wavelet of prototype 3 (and also for a wavelet of prototype 9), a part of the support would be located outside the patch, which is why these prototypes are inadmissible.

**Algorithm 5:** FITSIN

Input: Integer $k$ being 1, 2 or 3 and wavelet `w`
Output: 1, if `index` is the correct prototype, else 0

Set: `index` = `switchRules[w` $\rightarrow$ `protoType-1][`$k$`]`;
**for** $i = 0$ **to** `supportSize[index][1]` **do**
  **if** 0 = RELATIVENEIGHBOURS `(waveSupport[index][`$i$`])` **then**
    return 0;
  **end**
**end**
return 1;

It was already mentioned that the arrays `supportSize` and `waveSupport` are equipped with a second set of integers and additional characters, such that the complete information for both bases is stored within. Thus, as soon as the prototype of all wavelets is known, the array `support` and the pointer `*master` can be set as usual. For the current situation, where we switch between Haar wavelets and wavelets with three vanishing moments, the arrays are defined as

```
int* supportSize[10] = {
    (int[1]) {1},
    (int[2]) {4, 6}, (int[2]) {2, 6}, (int[2]) {2, 6},
    (int[2]) {0, 6}, (int[2]) {0, 6}, (int[2]) {0, 6},
    (int[2]) {0, 6}, (int[2]) {0, 6}, (int[2]) {0, 6}
};
```

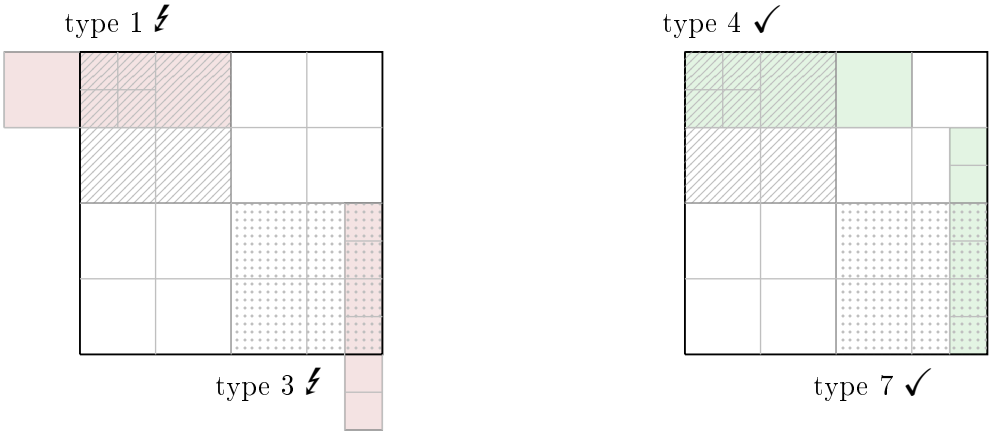



Figure 4.8: Illustration of the function FitsIn.

Likewise, the array `inheritance` is enhanced with the data for the newly appearing prototypes. It contains the following integers:

```
int** inherit[10] = {
    (int*[2]) { (int[3]) {1,2,3},   (int[3]) {0,0,0} },
    (int*[2]) { (int[4]) {1,1,1,1}, (int[4]) {0,1,2,3} },
    (int*[2]) { (int[4]) {2,3,3,2}, (int[4]) {0,0,1,1} },
    (int*[2]) { (int[4]) {2,3,3,2}, (int[4]) {0,0,1,1} },
    (int*[2]) { (int[4]) {4,1,1,4}, (int[4]) {0,1,2,3} },
    (int*[2]) { (int[4]) {1,5,5,1}, (int[4]) {0,1,2,3} },
    (int*[2]) { (int[4]) {6,7,3,2}, (int[4]) {0,0,1,1} },
    (int*[2]) { (int[4]) {6,7,3,2}, (int[4]) {0,0,1,1} },
    (int*[2]) { (int[4]) {2,3,9,8}, (int[4]) {0,0,1,1} },
    (int*[2]) { (int[4]) {2,3,9,8}, (int[4]) {0,0,1,1} }
};
```

**Remark 28.** We will complete this subsection with a final remark. Some routines rely on the fact that the support of the wavelets' descendants is always contained in the parents' support. One such example is the routine which computes the pattern for the matrix compression. In general, this condition is fulfilled. However, during the change of Haar wavelets to wavelets with three vanishing moments, this condition is violated, since the new wavelets are larger than the Haar wavelets which have been used before. This is a problem which cannot be solved within the wavelet archive, but has to be taken care of in the according routine, i.e. compress and predict.

## 4.3 The Adaptive Algorithm

Since the most important underlying structures are now explained, we can turn towards the details of the implementation. In this section, we will first give an overview of the work-flow of the program as well as the interaction and hierarchy between the routines. Figure 4.9 illustrates the control-flow diagram of the program.

### 4.3.1 Overview

The program starts by initialising the parametrisation, followed by building the underlying tree structures, as explained in Subsections 4.1.2 and 4.2.1. At the same time, we initialise the array `pointList`, as outlined in Algorithm 2, followed by finding and setting the neighbour elements. Furthermore, we compute the points, which are needed later for evaluating the potential. While the trees are initialised, they automatically become intertwined, as whenever a new wavelet is added to the wavelet tree, possibly new elements will be added, giving a link to the element tree. Vice versa, a new wavelet appears in the array of wavelets for the according element, giving a link back to the wavelet tree, too.

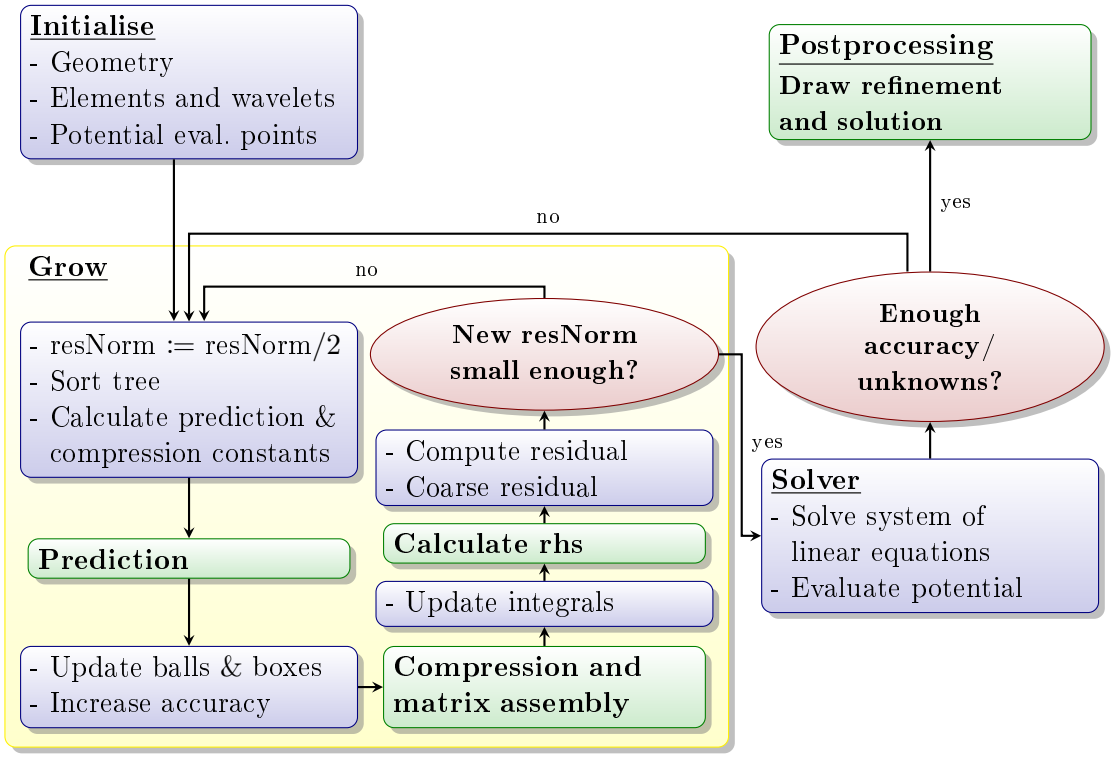



Figure 4.9: A schematic display of the control-flow diagram.

**Remark 29.** Despite the fact of having an element tree, the implementation does not rely on an underlying mesh. Indeed, it is a meshless method since the refinement of wavelets

does not depend on where we decide to refine elements. Whenever it is decided that new wavelet functions are activated, elements will be refined as a consequence. The only two exceptions to this are the assembly routine, which is able to refine elements for quadrature, and the routine for the evaluation of the potential.

As soon as the element tree and the wavelet tree are initialised, we will start a loop consisting of different routines, which we will subsume as `grow` in what follows. The basic goal of this loop is to grow the wavelet tree. Starting with an initial guess for the estimated norm of the residual, the routine will perform a prediction step which adds appropriate new wavelet functions. In a next step, the system matrix is assembled with respect to this extended set of basis functions. After calculating the discrete approximation to the right-hand side, we compute the new approximation to the residual. For this calculation, we need only a matrix-vector multiplication. The loop terminates as soon as the estimated norm falls below a certain threshold, after which we solve the system of linear equations and check again if the desired accuracy is met. If it is, the code terminates and, if desired, we can visualise the solution and the refinement of the geometry. Otherwise, we repeat the growing process. For solving the system of linear equations, we use the preconditioned conjugate gradient method [62] if the integral operator under consideration is symmetric and positive definite, which is the case for an integral equation featuring only the single layer operator (1.3). In the other cases, we use the (restarted) GMRES method [80]. That is, for example, for equations which feature the double layer operator (1.7), or for the system of linear equations arising from the Helmholtz equation.

During the next subsections, we will step-by-step address the routines which are outlined in Figure 4.9 and go into detail on their efficient implementation.

### 4.3.2 Layer Classification and Tree Sorting

In the process of growing the wavelet tree, we will first have to sort the tree. In Subsection 3.3.2, we already discussed the theoretical aspects of sorting and coarsening trees, whereas the current subsection focusses on the implementation. During the runtime of the algorithm, there are two situations where the wavelet tree needs to be sorted, namely for the prediction and for the coarsening of the residual at the end of the growing cycle. The way of sorting the tree hereby differs, dependent on which of the two situations occurs.

For prediction as well as for coarsening, we first have to arrange the wavelets into layers, in order to fulfil the estimate (3.21). Recall that the theoretical aspect of doing so has already been mentioned in Subsection 3.3.3. In the implementation, we store the information on the wavelet layer in the integer `layer`. In order to be even able to collate the wavelets into different layers, we need two additional doubles `down` and `up` to sort the tree in advance. The doubles `down` and `up` correspond to the functionals $e$ and $\tilde{e}$, respectively, encountered in Subsection 3.3.2. The difference between prediction and coarsening lies in the way of computing the coefficients `down` and `up`, since we calculate these coefficients with respect

to a different norm:

$$\begin{aligned} \texttt{sort}(\boldsymbol{v}, \|\cdot\|_s) \quad &\text{with } \|\boldsymbol{v}\|_q^2 \sim \sum_{\boldsymbol{\lambda}\in\boldsymbol{v}} v_{\boldsymbol{\lambda}}^2 \, |\boldsymbol{A}_{\boldsymbol{\lambda},\boldsymbol{\lambda}}|, \\ \texttt{sort}(\boldsymbol{v}, \|\cdot\|_{-s}) \quad &\text{with } \|\boldsymbol{v}\|_{-q}^2 \sim \sum_{\boldsymbol{\lambda}\in\boldsymbol{v}} \frac{v_{\boldsymbol{\lambda}}^2}{|\boldsymbol{A}_{\boldsymbol{\lambda},\boldsymbol{\lambda}}|}. \end{aligned} \tag{4.3}$$

Here, each $\boldsymbol{\lambda} = (i, j, \boldsymbol{k}, \iota)$ is associated with an active wavelet $\psi_{\boldsymbol{\lambda}}$. The coefficient $v_{\boldsymbol{\lambda}}$ is the $\boldsymbol{\lambda}$-th coefficient of the vector $\boldsymbol{v}$ whose index set has tree structure. The term $\boldsymbol{A}_{\boldsymbol{\lambda},\boldsymbol{\lambda}}$ is the diagonal entry of the system matrix. As mentioned in Subsection 3.3.2, we will first run recursively through the tree bottom-up, starting with calculating the coefficient for each leaf wavelet. The procedure performing this step is outlined in Algorithm 6.

**Algorithm 6:** void UP(w)

Set: w → up = 0;

**if** w → child ≠ NULL **then**
  **for** $i = 0$ **to** noChildren[w] **do**
    UP(w → child[$i$]);
    Update: w → up += w → child[$i$] → up;
  **end**
**end**

Update: w → up (according to the underlying norm);
Update: wNorm (according to the underlying norm);

Whenever we later refer to sorting the tree, we will make clear which norm has been used. After Algorithm 6 terminates, we have a hierarchy on the wavelet tree, since all parent wavelets have a larger coefficient than each of their children. The second algorithm, outlined in Algorithm 7, runs through the tree top-down. It preserves the hierarchy among the wavelets but, in addition, the generated coefficients are equal for all siblings. This makes sense, since all children of one wavelet are of equal importance. The value $q$, which is initialised first, is according to equation (3.13) and also for the root node it holds that $\tilde{e}(\boldsymbol{\lambda}) = e(\boldsymbol{\lambda})$.

**Algorithm 7:** void DOWN(w)

Initialise: q = w → down · ($\sum_i$ w → child[$i$] → up )/(w → up + w → down);

**if** w → child ≠ NULL **then**
  **for** $i = 0$ **to** noChildren[w] **do**
    Set: w → child[$i$] → down = q;
    DOWN(w → child[$i$]);
  **end**
**end**

After Algorithm 7 terminates, each active wavelet $\psi_{\boldsymbol{\lambda}}$ is equipped with a coefficient `down`. This coefficient is a global representative inside the wavelet tree of either the residuum or the solution, cf. Subsection 3.3.2. Finally, we sort the wavelet tree according to the magnitude of these coefficients. Thus, the question arises which sorting algorithm should be used best.

Let us briefly address pros and cons of the two most common known sorting algorithms Quicksort and merge sort. No sorting algorithm is able to perform in linear complexity. Merge sort has complexity $\mathcal{O}(N \log(N))$ in the best case, average case and worst case. Quicksort has the same best and average complexity, but it is of quadratic worst case complexity, see [69]. If we want a sorting routine which performs with linear complexity, we have to use a quasi-sorting instead, for example bucket sort. With quasi-sorting we mean that the coefficients are sorted into buckets according to their order of magnitude, i.e. without further sorting the coefficients inside the buckets. This does not result in a completely sorted tree, but is sufficient for our purpose.

In our implementation, however, we do not use bucket sort. We use a slightly better version of merge sort, by using so-called runs. The standard implementation divides the array dyadically until only single values remain to be compared and arranged in the correct order. Then, each pair of sorted values is merged together with another pair, resulting in an array with four correctly sorted values. This is done recursively until the complete array is sorted. If we know in advance that large parts of the array are sorted natively, which is the case for our tree, we do not have to subdivide the array until only one value remains. These pre-sorted parts of the tree (runs) have to be found in advance, which normally requires one extra step to use this version of merge sort. Fortunately, we know from the beginning that parent wavelets will end up having a larger coefficient than their children. Thus, we collect the information about the runs while executing Algorithm 7.

**Algorithm 8:** Layer classification.

Wavelets are now sorted in the array `sortVec`;
Set: `err` $=$ `ctr` $= 0$;
Set: `thresh`;
**for** $i =$ `maxLayer` **to** $1$ **do**
    Update: `thresh`;
    **while** `thresh` $>$ `err` **do**
        Update: `err +=` $(\texttt{sortVec[ctr]} \to \texttt{sol})^2 \cdot \texttt{sortVec[ctr]} \to \texttt{diag}$;
        Set: `sortVec[ctr]` $\to$ `layer` $= i$;
        Set: `++ctr`;
    **end**
**end**
**for** $j =$ `ctr` **to** `nWavelets` **do** Set: `sortVec[ctr]` $\to$ `layer` $= 0$;

With the sorted tree at hand, we can set the layer of each wavelet according to Algorithm

8, such that the estimate (3.21) holds. The arrangement into layers $\Delta_j$ is not only useful for prediction and coarsening, but also for compression and assembling the system matrix.

### 4.3.3 Compress and Predict

After the tree is sorted via $\texttt{sort}(\boldsymbol{v}, \|\cdot\|_q)$ according to equation (4.3), the next routine called in Grow is the routine for prediction. The input $\boldsymbol{v}$, which is to be sorted, is the solution to the system of linear equations from the previous step. After the tree is sorted and the arrangement into layers is completed, we will start the prediction routine. As prediction and compression work very similarly, we will focus on both routines in this subsection with the main focus on the compression.

In compress and predict, we need to be able to distinguish between wavelets that have been in the tree before entering the growing loop and wavelets that are newly added. For this reason, each wavelet has a flag `activity`, which can take three different values:

$$\texttt{wavelet} \to \texttt{activity} = \begin{cases} 0, & \text{if the wavelet is inactive,} \\ 1, & \text{if the wavelet is activated permanently,} \\ 2, & \text{if the wavelet is activated for test purposes.} \end{cases}$$

Every time we enter the growing routine after having solved the system of linear equations in the main routine, each active wavelet in the tree has its activity set to 1. The routines for prediction and compression are recursive functions, which check how a new wavelet interacts with the other wavelets. Whenever we decide to add a new wavelet in the prediction step, we assign the value 2. The wavelet can either keep this value, or it will be inactivated again by coarsening (see Figure 4.9). After growing has been completed, we set the value for all wavelets with activity 2 to activity 1, meaning that they are permanently added.

Before we proceed with the implementation in detail, we shall recall some notation from Subsection 2.5. We write $\psi_{\boldsymbol{\lambda}}$ with $\boldsymbol{\lambda} = (i, j, \boldsymbol{k}, \iota)$, for a wavelet $\psi^{\Gamma_i}_{j,\boldsymbol{k},\iota}$ of level $j$ with prototype $\iota$ on the $i$-th patch $\Gamma_i$, where the index $\boldsymbol{k} = (k_1, k_2)$ indicates the location of its master element inside the patch. For convenience, we use again the abbreviation $|\boldsymbol{\lambda}| = j$ for the level. With $\hat{j} := j + \log_2(\alpha_j)$, the compression pattern has been identified in [87]. The entries of $\boldsymbol{A}_j$ (see Definition 25) are obtained from $\boldsymbol{A}$ by setting

$$(\boldsymbol{A}_j)_{\boldsymbol{\lambda},\boldsymbol{\lambda}'} = \begin{cases} 0, \text{ if } ||\boldsymbol{\lambda}| - |\boldsymbol{\lambda}'|| > \hat{j}, \\ 0, \text{ if } \operatorname{dist}(\operatorname{supp}(\psi_{\boldsymbol{\lambda}}), \operatorname{supp}(\psi_{\boldsymbol{\lambda}'})) > C_1^{\hat{j}}(j, j'), \\ 0, \text{ if } |\boldsymbol{\lambda}| - |\boldsymbol{\lambda}'| > \frac{\hat{j}}{2} \text{ and } 2^{|\boldsymbol{\lambda}'|}\operatorname{dist}(\operatorname{supp}(\psi_{\boldsymbol{\lambda}}), \operatorname{sing\,supp}(\psi_{\boldsymbol{\lambda}'})) > C_2^{\hat{j}}(j, j'), \\ 0, \text{ if } |\boldsymbol{\lambda}'| - |\boldsymbol{\lambda}| > \frac{\hat{j}}{2} \text{ and } 2^{|\boldsymbol{\lambda}|}\operatorname{dist}(\operatorname{sing\,supp}(\psi_{\boldsymbol{\lambda}}), \operatorname{supp}(\psi_{\boldsymbol{\lambda}'})) > C_2^{\hat{j}}(j, j'), \\ \langle \mathcal{A}\psi_{\boldsymbol{\lambda}}, \psi_{\boldsymbol{\lambda}'} \rangle, \text{ else.} \end{cases} \tag{4.4}$$

The cut-off parameters $C_1^{\hat{j}}(j,j')$ and $C_2^{\hat{j}}(j,j')$ are given by

$$\begin{aligned} C_1^{\hat{j}}(j,j') &:= a\ 2^{-\min\{j,j'\}} \max\left\{1,\ 2^{b(\frac{\hat{j}}{2}-|j-j'|)}\right\}, \\ C_2^{\hat{j}}(j,j') &:= a\ \max\left\{\gamma_{|j-j'|-\frac{\hat{j}}{2}}\, 2^{\hat{j}-2|j-j'|}, 2^{-|j-j'|}\right\}. \end{aligned} \tag{4.5}$$

Here, $a \geq 1$ and $1/4 \leq b < 1/2$ are suitable fixed parameters and $(\gamma_n)_{n\in\mathbb{N}}$ is a polynomially decreasing sequence such that $\sum_{n\in\mathbb{N}} \gamma_n < \infty$. With $\operatorname{dist}(\operatorname{supp}(\psi_{\boldsymbol{\lambda}}), \operatorname{supp}(\psi_{\boldsymbol{\lambda}'}))$ we denote the distance between the two wavelets $\psi_{\boldsymbol{\lambda}}$ and $\psi_{\boldsymbol{\lambda}'}$. To be more precise, we mean the three-dimensional distance between the convex hulls of the wavelets on the surface, which is calculated by using the bounding boxes of the wavelets. Also, we recall the singular support of a wavelet $\psi_{\boldsymbol{\lambda}}$, denoted with $\operatorname{sing\,supp}(\psi_{\boldsymbol{\lambda}})$, which is the set of all points on which the wavelet is not smooth. To compute this distance, we consider the distance between the elements in their supports on the unit square, as the smaller wavelet is located inside the larger wavelet.

The constant $a \geq 1$, which appears in the cut-off parameter, is the so-called bandwidth parameter. It has an influence on the sparsity of the matrix, where a lager $a$ results in a more dense matrix. In our simulations, we choose $a = 2.5$, $b = 0.5$ for prediction, $b = 0.75$ for compression and $\gamma_n = 1/(1+n)^2$. In case we consider the Helmholtz equation, the bandwidth parameter $a$ becomes dependent on the wavenumber $\kappa$, see [65] for details. In particular, we choose $a = a \cdot (1 + (\kappa/4))$. The consequence is that the matrix gets even more dense.

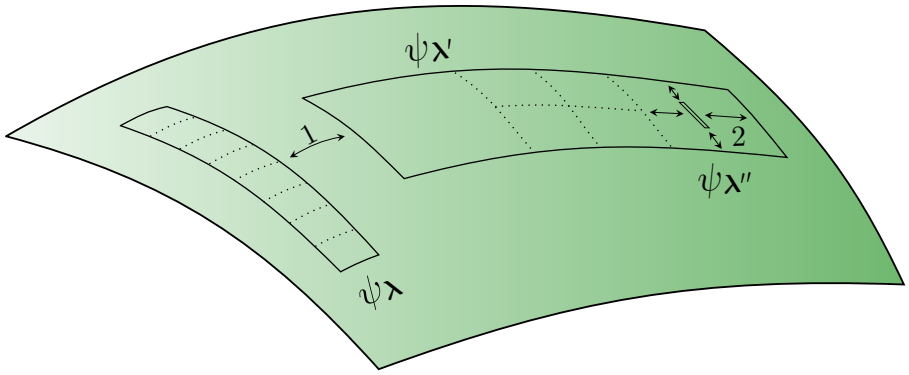



Figure 4.10: Illustration of a situation which is affected by the first compression (labelled with 1) and by the second compression (labelled with 2).

The second condition in equation (4.4) amounts to the first compression, whereas the third and forth conditions in equation (4.4) amount to the second compression. The first compression applies to two wavelets with disjoint supports and makes use of the distance between them. The second compression applies to two wavelets for which the first compression cannot be applied. It can be used when they have a large level difference and one wavelet lies in the smooth part of the other, see Figure 4.10 for an illustration.

A useful property of these compression rules is that we can draw conclusions for neglecting matrix entries of children wavelets by knowing whether the first or second compression applies to their parents. Suppose that we have given the wavelets $\psi_{\boldsymbol{\lambda}}$ and $\psi_{\boldsymbol{\lambda}'}$ which have a disjoint support. Furthermore, let $\tilde{\boldsymbol{\lambda}}$ be in the set of 4-tuples $(i, j+1, \tilde{\boldsymbol{k}}, \tilde{\iota})$ such that $\psi_{\tilde{\boldsymbol{\lambda}}}$ is a child of $\psi_{\boldsymbol{\lambda}}$. Analogously, let $\tilde{\boldsymbol{\lambda}}'$ be in the set of 4-tuples $(i', j'+1, \tilde{\boldsymbol{k}}', \tilde{\iota}')$ such that $\psi_{\tilde{\boldsymbol{\lambda}}'}$ is a child of $\psi_{\boldsymbol{\lambda}'}$. Let us stress two properties for the involved cut-off parameter $C_1^{\hat{j}}(j, j')$. First, the cut-off parameter $C_1^{\hat{j}}(j, j')$ is symmetric with respect to the levels $j$ and $j'$, i.e. $C_1^{\hat{j}}(j, j') = C_1^{\hat{j}}(j', j)$. Second, for the cut-off parameters $C_1^{\hat{j}}(\tilde{j}, j')$ and $C_1^{\hat{j}}(j, \tilde{j}')$ of the children $\psi_{\tilde{\boldsymbol{\lambda}}}$ and $\psi_{\tilde{\boldsymbol{\lambda}}'}$ (i.e. $\tilde{j} = j+1$ and $\tilde{j}' = j'+1$) there holds that $C_1^{\hat{j}}(\tilde{j}, \tilde{j}') \leq C_1^{\hat{j}}(\tilde{j}, j')$, $C_1^{\hat{j}}(j, \tilde{j}') \leq C_1^{\hat{j}}(j, j')$.

We can thus compute the distance between $\psi_{\boldsymbol{\lambda}}$ and $\psi_{\boldsymbol{\lambda}'}$ and check if the first compression applies. If the answer is yes, then we have the following lemma cf. [54]:

**Lemma 30.** If $\operatorname{dist}(\operatorname{supp}(\psi_{\boldsymbol{\lambda}}), \operatorname{supp}(\psi_{\boldsymbol{\lambda}'})) > C_1^{\hat{j}}(j, j')$, then there holds

$$\operatorname{dist}(\operatorname{supp}(\psi_{\tilde{\boldsymbol{\lambda}}}), \operatorname{supp}(\psi_{\boldsymbol{\lambda}'})) > C_1^{\hat{j}}(\tilde{j}, j'),$$

$$\operatorname{dist}(\operatorname{supp}(\psi_{\boldsymbol{\lambda}}), \operatorname{supp}(\psi_{\tilde{\boldsymbol{\lambda}}'})) > C_1^{\hat{j}}(j, \tilde{j}'),$$

$$\operatorname{dist}(\operatorname{supp}(\psi_{\tilde{\boldsymbol{\lambda}}}), \operatorname{supp}(\psi_{\tilde{\boldsymbol{\lambda}}'})) > C_1^{\hat{j}}(\tilde{j}, \tilde{j}'),$$

for all children $\psi_{\tilde{\boldsymbol{\lambda}}}$ of $\psi_{\boldsymbol{\lambda}}$ and all children $\psi_{\tilde{\boldsymbol{\lambda}}'}$ of $\psi_{\boldsymbol{\lambda}'}$.

This lemma holds true due to the monotonicity of the cut-off parameters as explained earlier and due to the fact that the support of a wavelet's child is always contained in the parent's support, i.e. $\operatorname{supp}(\psi_{\tilde{\boldsymbol{\lambda}}}) \subset \operatorname{supp}(\psi_{\boldsymbol{\lambda}})$ for all children. Thus, whenever the first compression applies for $(\boldsymbol{A}_j)_{\boldsymbol{\lambda},\boldsymbol{\lambda}'}$, we can neglect the matrix entries $(\boldsymbol{A}_j)_{\tilde{\boldsymbol{\lambda}},\boldsymbol{\lambda}'}, (\boldsymbol{A}_j)_{\boldsymbol{\lambda},\tilde{\boldsymbol{\lambda}}'}$ and $(\boldsymbol{A}_j)_{\tilde{\boldsymbol{\lambda}},\tilde{\boldsymbol{\lambda}}'}$ as well.

A similar rule is valid for the second compression. Suppose that the two wavelets $\psi_{\boldsymbol{\lambda}}$ and $\psi_{\boldsymbol{\lambda}'}$ are located on the same patch $i$, i.e. $\boldsymbol{\lambda} = (i, j, \boldsymbol{k}, \iota)$ and $\boldsymbol{\lambda}' = (i, j', \boldsymbol{k}', \iota')$. Furthermore, assume that the wavelet $\psi_{\boldsymbol{\lambda}'}$ is located inside the wavelet $\psi_{\boldsymbol{\lambda}}$, which especially implies $j' \gg j$. The cut-off parameter, associated with the second compression, is again symmetric with respect to the input levels, i.e. $C_2^{\hat{j}}(j, j') = C_2^{\hat{j}}(j', j)$. In addition, there holds $C_2^{\hat{j}}(j, j'+1) \leq C_2^{\hat{j}}(j, j')$. Thus, there holds the following lemma cf. [54]:

**Lemma 31.** If $\operatorname{dist}(\operatorname{supp}(\psi_{\boldsymbol{\lambda}'}), \operatorname{sing\,supp}(\psi_{\boldsymbol{\lambda}})) > C_2^{\hat{j}}(j, j')$ then there holds that

$$\operatorname{dist}(\operatorname{supp}(\psi_{\tilde{\boldsymbol{\lambda}}'}), \operatorname{sing\,supp}(\psi_{\boldsymbol{\lambda}})) > C_2^{\hat{j}}(j, j'+1),$$

for all children $\psi_{\tilde{\boldsymbol{\lambda}}'}$ of $\psi_{\boldsymbol{\lambda}'}$.

Consequently, if the matrix entry $(\boldsymbol{A}_j)_{\boldsymbol{\lambda},\boldsymbol{\lambda}'}$ can be neglected, then the entry $(\boldsymbol{A}_j)_{\boldsymbol{\lambda},\tilde{\boldsymbol{\lambda}}'}$ can also be neglected.

From the above discussion, we easily see that the first compression and the second compression have a different inheritance behaviour for wavelet children. Since our implementation uses a recursive routine for the compression and the prediction, it is of advantage being able to decide which of the two compression rules apply. The routine CRITERION, outlined in Algorithm 9, checks whether a matrix entry can be neglected and, if yes, why. It has return values in the set $\{0, 1, 2, 3, 4\}$. If the matrix entry has to be calculated, the routine returns the value 0. If the matrix entry can be neglected, it returns a value between 1 and 4, helping to decide how the recursion has to proceed.

**Algorithm 9:** CRITERION

Input: Wavelets `w1`, `w2`;

Short tests:
**if** Both wavelets are inactive **then** return 1;
**if** The level difference is too large **then** return 2;
**if** The distance of the bounding boxes on the surface is too large **then** return 3;
**if** The level difference is too small **then** return 0;

If all above tests failed, we have to run through the support:
Set: $l_1 =$ `w1` $\rightarrow$ `level`, $l_2 =$ `w2` $\rightarrow$ `level`;
**for** $i = 0$ **to** `suppSize[w1` $\rightarrow$ `protoType][kind[w1]]` **do**
  **for** $j = 0$ **to** `suppSize[w2` $\rightarrow$ `protoType][kind[w2]]` **do**
    **if** `w1` $\rightarrow$ `patch = w2` $\rightarrow$ `patch` **then**
      Calculate: distance $d_2$ on unit square;
      **if** $d_2 < \hat{C}_2^j(l_1, l_2)$ **then** return 0;
    **else**
      Calculate: $d_3 =$ DIST3D(`w1` $\rightarrow$ `support[`$i$`], w2` $\rightarrow$ `support[`$j$`]`);
      **if** $d_3 < \hat{C}_2^j(l_1, l_2)$ **then** return 0;
    **end**
  **end**
**end**
return 4;

We observe that there are four different reasons why a matrix entry may be neglected. Algorithm 9 first performs four short tests, already ruling out a large amount of wavelets, before having to check distances of the supporting elements or calculating three-dimensional distances. Each return value has a different meaning and a different consequence:

0: The matrix entry has to be calculated and is set in the compression pattern.

1: Both wavelets are inactive and, thus, the children are not active either. The recursion can stop for both wavelets.

2: The level difference is too large. The recursion can stop for the wavelet on the finer refinement scale.

3: The first compression applies. Thus, the recursion can stop unconditionally.

4: The second compression applies. The recursion can stop for the wavelet on the fine refinement scale.

The two routines compression and prediction are initialised for each scaling function. Moreover, we call these routines in a way such that we can guarantee that the level of the first wavelet is always larger or equal to the level of the second wavelet. This condition makes it easy to call the routine in a further recursion. After we started the prediction or the compression, respectively, the routine recursively runs through the tree by calling itself for the children until one of the stopping criteria is met. In Algorithm 10, we see the outline of how the recursion works for prediction and compression. Prediction and compression are based on a very similar idea for the recursion. The difference is the ability of the prediction routine (highlighted in red) to further call the routine for wavelet refinement if the children are not yet part of the tree. The routine for compression will stop if the recursive call for its children cannot be performed.

The recursive call of the prediction and compression routine returns the value which has been computed inside the function CRITERION. Its result affects how the recursion has to proceed. Thus, this result can directly be used inside the recursive call. Recall that an even return value of the function CRITERION means that the matrix entry cannot be neglected, in which case the recursion has to continue with the children of the wavelet. Whenever the criterion returns the value zero, we will add the pair of wavelet indices to the compression pattern, leaving us with a complete set of pairs, for which we need to determine the matrix entry by quadrature. Before we discuss this, we will briefly mention the structure, which is used to represent the compressed matrix.

### 4.3.4 Structure `Sparse`

Since the system of linear equations emerging from our adaptive wavelet method is quasi-sparse, we use the compressed row storage format to work with the system matrix. We need thus the following ingredients, which are stored in the structure `sparse`:

```
double  **value;
int     **index, *rowNumber, *maxRowNumber, noRows, noColumns;
```

Therein, the variables `noRows` and `noColumns` denote the dimension of the system matrix. In our case, these dimensions are the same, since we work always with a quadratic system matrix. For each row of the matrix, there is an array associated with it. This array contains the matrix entries, stored in `value`, and the index where it is found inside the according row is stored in `index`. The variables `rowNumber` and `maxRowNumber` indicate how many entries are stored in the row and how much memory is allocated, respectively. Figure 4.11 illustrates the data structure.

**Algorithm 10:** Recursive call of prediction and compression.

Input: Wavelets `w1`, `w2`;
Output: c given by the output of the routine CRITERION;

```
c = CRITERION(w1, w2);
if c = 0 then
    Add the wavelet indices to the compression pattern;
else
    return c;
end
if (w1 → level = w2 → level) then
    if (w1 ≠ w2) then
        for i = 0 to noChildren do
            r1[i] = COMPRESS/PREDICT(w1 → child[i], w2);
        end
        for j = 0 to noChildren do
            r2[j] = COMPRESS/PREDICT(w2 → child[j], w1);
        end
        for i, j = 0 to noChildren do
            if r1[i] and r2[j] are even then
                COMPRESS/PREDICT(w1 → child[i], w2 → child[j]);
            end
        end
    else // w1 = w2
        for i = 0 to noChildren do
            r1[i] = COMPRESS/PREDICT(w1 → child[i], w1);
        end
        for i, j = 0 to noChildren do
            if r1[i] and r1[j] are even then
                COMPRESS/PREDICT(w1 → child[i], w1 → child[j]);
            end
        end
    end
else // w1 → level != w2 → level
    for i = 0 to noChildren do
        COMPRESS/PREDICT(w1 → child[i], w2);
    end
end
```

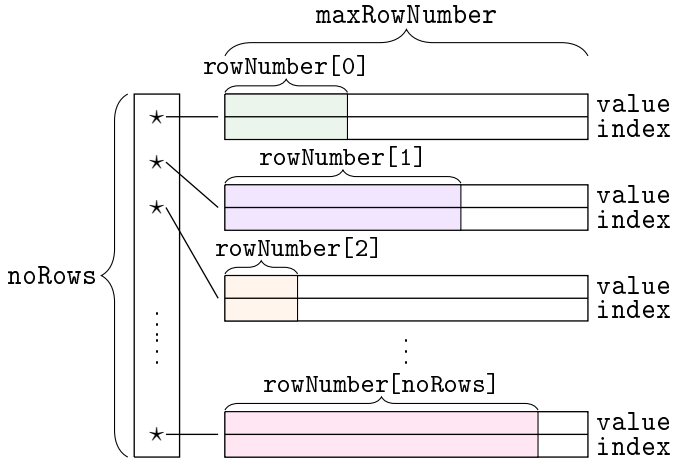



Figure 4.11: Illustration of the sparse data structure.

The compression routine determines all indices, which have to be set in the array `index`. During the compression process, the array `value` is not modified. Whenever the criterion routine returns zero, we set a new entry in the compression pattern for the according wavelet index. New indices are then inserted into the array `index` with respect to their modulus. After compression is complete, we have thus at hand all the required indices in increasing order, as well as the `rowNumber` for each row of the matrix. The pattern is finally completed by allocating the array `value`, using the correct size `rowNumber[i]` for all columns, as well as reallocating the array of indices by this length.

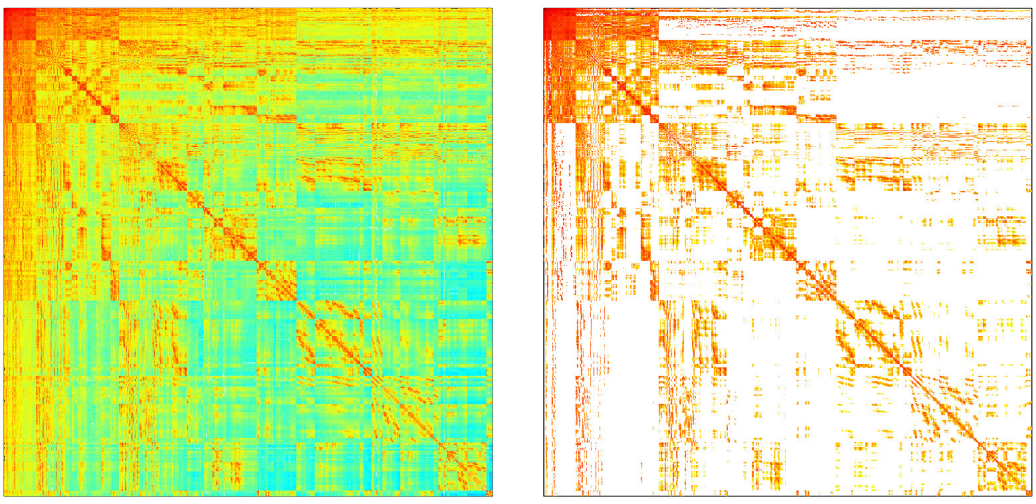


Figure 4.12: Original matrix (left) and compressed matrix (right).

Figure 4.12 shows us the system matrices emerged from the implementation. These particular matrices originate from our adaptive implementation of the single layer operator on the Fichera vertex for 2946 unknowns. In the left picture, we see the full matrix, i.e.

with all matrix entries calculated. In the right picture, we see the compressed matrix for the compression parameter $a = 2$.

### 4.3.5 Apply

After the compression pattern has been identified, the relevant matrix entries have to be calculated. To this end, let us first introduce the theoretical aspects of calculating the matrix entries by quadrature, followed by the details on the implementation. All matrix entries marked by the compression will be calculated here, the indices of which have been stored in the sparse structure in advance.

For the matrix assembly, we have to calculate integrals of the following form

$$\boldsymbol{A}_{\boldsymbol{\lambda},\boldsymbol{\lambda}'} = \langle \mathcal{A}\psi_{\boldsymbol{\lambda}'}, \psi_{\boldsymbol{\lambda}} \rangle = \int_\Gamma \int_\Gamma k(\boldsymbol{x},\boldsymbol{y})\, \psi_{\boldsymbol{\lambda}'}(\boldsymbol{y}) \psi_{\boldsymbol{\lambda}}(\boldsymbol{x})\, \mathrm{d}\sigma_{\boldsymbol{x}}\, \mathrm{d}\sigma_{\boldsymbol{y}}$$

with $k(\boldsymbol{x},\boldsymbol{y})$ being the kernel function of the according integral operator. The index $\boldsymbol{\lambda} = (i, j, \boldsymbol{k}, \iota)$ for a wavelet $\psi_{\boldsymbol{\lambda}}$ encodes again the patch $i$, the level $j$, the location $\boldsymbol{k}$, and the prototype $\iota$. Transporting the patch on the surface to the unit square by the liftings $\boldsymbol{\gamma}_i \colon \square \to \Gamma_i$ gives

$$\boldsymbol{A}_{\boldsymbol{\lambda},\boldsymbol{\lambda}'} = \int_\square \int_\square k_{i,i'}(\boldsymbol{s},\boldsymbol{t})\, \psi_{\boldsymbol{\lambda}'}(\boldsymbol{\gamma}_{i'}(\boldsymbol{t})) \psi_{\boldsymbol{\lambda}}(\boldsymbol{\gamma}_i(\boldsymbol{s}))\, \mathrm{d}\boldsymbol{s}\, \mathrm{d}\boldsymbol{t} \tag{4.6}$$

where $k_{i,i'}(\boldsymbol{s},\boldsymbol{t}) \coloneqq k(\boldsymbol{\gamma}_i(\boldsymbol{s}), \boldsymbol{\gamma}_{i'}(\boldsymbol{t}))\mu_i(\boldsymbol{s})\mu_{i'}(\boldsymbol{t})$ is the transported kernel which contains the surface measures $\mu_i(\boldsymbol{s})$ and $\mu_{i'}(\boldsymbol{t})$ and $\boldsymbol{s} = \boldsymbol{\gamma}_i^{-1}(\boldsymbol{x})$, $\boldsymbol{t} = \boldsymbol{\gamma}_{i'}^{-1}(\boldsymbol{y})$. Inserting the representation (4.2) for any wavelet $\psi^\square_{j,\boldsymbol{k},\iota}(\boldsymbol{s})$ on the unit square, leads to

$$\begin{aligned} \boldsymbol{A}_{\boldsymbol{\lambda},\boldsymbol{\lambda}'} &= \sum_{(j,\boldsymbol{k})\in\mathcal{I}} \sum_{(j',\boldsymbol{k}')\in\mathcal{I}'} \int_{\square_{j,\boldsymbol{k}}} \int_{\square_{j',\boldsymbol{k}'}} k_{i,i'}(\boldsymbol{s},\boldsymbol{t})\, \phi^\square_{j',\boldsymbol{k}'}(\boldsymbol{t})\, w_{\boldsymbol{k}',\iota'}\, \phi^\square_{j,\boldsymbol{k}}(\boldsymbol{s})\, w_{\boldsymbol{k},\iota}\, \mathrm{d}\boldsymbol{s}\, \mathrm{d}\boldsymbol{t} \\ &= \sum_{(j,\boldsymbol{k})\in\mathcal{I}} \sum_{(j',\boldsymbol{k}')\in\mathcal{I}'} 2^{j+j'}\, w_{\boldsymbol{k}',\iota'}\, w_{\boldsymbol{k},\iota} \int_{\square_{j,\boldsymbol{k}}} \int_{\square_{j',\boldsymbol{k}'}} k_{i,i'}(\boldsymbol{s},\boldsymbol{t})\, \mathrm{d}\boldsymbol{s}\, \mathrm{d}\boldsymbol{t}. \end{aligned} \tag{4.7}$$

The factor $2^{j+j'}$, which appears in the last equality, comes from our $L^2$-normalised piecewise constant ansatz functions $\phi^\square_{j,\boldsymbol{k}}$. Notice that we reduced the calculation of the matrix coefficient to the calculation of the sum of interactions between pairs of elements. With equation (4.7) given, we will now discuss how to approximate the appearing integrals.

### 4.3.6 Gauss Quadrature and Error

For the approximation of the integrals appearing in equation (4.7), we use four-dimensional tensor product Gauss-Legendre quadrature rules. For our purpose, we need to be able to

approximate the following four-dimensional integral

$$I_{(\square_{j,\boldsymbol{k}}\times\square_{j',\boldsymbol{k}'})}[f] = 2^{j+j'}\int_{\square_{j,\boldsymbol{k}}}\int_{\square_{j',\boldsymbol{k}'}} f(\boldsymbol{s},\boldsymbol{t})\,\mathrm{d}\boldsymbol{s}\,\mathrm{d}\boldsymbol{t}.$$

To that end, we use the four-dimensional $(n,n')$-point tensor product Gauss-Legendre quadrature rule

$$Q^{(n,n')}_{(\square_{j,\boldsymbol{k}}\times\square_{j',\boldsymbol{k}'})}[f] = \sum_{l=1}^{n^2}\sum_{l'=1}^{n'^2}\boldsymbol{\omega}_l\,\boldsymbol{\omega}_{l'}\,f(\boldsymbol{\xi}_l,\boldsymbol{\xi}_{l'}). \tag{4.8}$$

There holds the following estimate for the quadrature error, see [54]:

**Lemma 32.** The error which is produced by the four-dimensional Gauss-Legendre quadrature rule is bounded by

$$\begin{aligned}\left|I_{(\square_{j,\boldsymbol{k}}\times\square_{j',\boldsymbol{k}'})}[f] - Q^{(n,n')}_{(\square_{j,\boldsymbol{k}}\times\square_{j',\boldsymbol{k}'})}[f]\right| &\\ \lesssim 2^{-(j+j')}\Bigg\{&\frac{2^{-2n(j+2)}}{(2n)!}\left[\max_{\substack{\boldsymbol{z}\in\square_{j,\boldsymbol{k}}\\ \boldsymbol{z}'\in\square_{j',\boldsymbol{k}'}}}\left|\frac{\partial^{2n}f(\boldsymbol{z},\boldsymbol{z}')}{\partial z_1^{2n}}\right| + \max_{\substack{\boldsymbol{z}\in\square_{j,\boldsymbol{k}}\\ \boldsymbol{z}'\in\square_{j',\boldsymbol{k}'}}}\left|\frac{\partial^{2n}f(\boldsymbol{z},\boldsymbol{z}')}{\partial z_2^{2n}}\right|\right]\\ &+\frac{2^{-2n'(j'+2)}}{(2n')!}\left[\max_{\substack{\boldsymbol{z}\in\square_{j,\boldsymbol{k}}\\ \boldsymbol{z}'\in\square_{j',\boldsymbol{k}'}}}\left|\frac{\partial^{2n'}f(\boldsymbol{z},\boldsymbol{z}')}{\partial z_1'^{2n'}}\right| + \max_{\substack{\boldsymbol{z}\in\square_{j,\boldsymbol{k}}\\ \boldsymbol{z}'\in\square_{j',\boldsymbol{k}'}}}\left|\frac{\partial^{2n'}f(\boldsymbol{z},\boldsymbol{z}')}{\partial z_2'^{2n'}}\right|\right]\Bigg\}.\end{aligned}$$

The integrals in (4.7) feature the kernel function. Recalling the Definition 24 for a kernel to be analytically standard of order $2s$ gives us immediately a bound for the partial derivatives of the kernel. We have the following lemma, see e.g. [59,84]:

**Lemma 33.** Let the diffeomorphisms $\boldsymbol{\gamma}_i\colon\square\to\Gamma_i$ be analytical for all $i=1,\dots,M$. Then, the kernels of the single layer operator (1.3), the double layer operator (1.7) and its adjoint (1.10) are analytically standard of the corresponding operator order.

Now, we want to describe the error which arises from applying the four-dimensional tensor product Gauss-Legendre quadrature rule to the four-dimensional integral featuring the transported kernel:

$$I_{(\square_{j,\boldsymbol{k}}\times\square_{j',\boldsymbol{k}'})}[k_{i,i'}(\boldsymbol{s},\boldsymbol{t})] = 2^{j+j'}\int_{\square_{j,\boldsymbol{k}}}\int_{\square_{j',\boldsymbol{k}'}} k_{i,i'}(\boldsymbol{s},\boldsymbol{t})\,\mathrm{d}\boldsymbol{s}\,\mathrm{d}\boldsymbol{t}.$$

Combining Lemma 32 with estimate (3.16) gives the following result.

**Proposition 34.** Let the kernel $k(\boldsymbol{x},\boldsymbol{y})$ be analytically standard of order $2s$. Assume that there holds $\operatorname{dist}(\Gamma_{i,j,\boldsymbol{k}},\Gamma_{i',j',\boldsymbol{k}'})>0$, where $\Gamma_{i,j,\boldsymbol{k}}=\boldsymbol{\gamma}_i(\square_{j,\boldsymbol{k}})$ with $\square_{j,\boldsymbol{k}}$ the element of level

$j$ at location $\boldsymbol{k} = (k_1, k_2)$ of the unit square. Then, the quadrature error is bounded by

$$\begin{aligned}\left|I_{(\square_{j,\boldsymbol{k}}\times\square_{j',\boldsymbol{k}'})}[k_{i,i'}(\boldsymbol{s},\boldsymbol{t})] - Q^{(n,n')}_{(\square_{j,\boldsymbol{k}}\times\square_{j',\boldsymbol{k}'})}[k_{i,i'}(\boldsymbol{s},\boldsymbol{t})]\right| \\ \lesssim 2^{-(j+j')}\left[\left(\frac{2^{-j}}{4r}\right)^{2n} \operatorname{dist}(\Gamma_{i,j,\boldsymbol{k}},\Gamma_{i',j',\boldsymbol{k}'})^{-(2+2s+2n)}\right. \\ \left. + \left(\frac{2^{-j'}}{4r}\right)^{2n'} \operatorname{dist}(\Gamma_{i,j,\boldsymbol{k}},\Gamma_{i',j',\boldsymbol{k}'})^{-(2+2s+2n')}\right].\end{aligned} \tag{4.9}$$

When using the four-dimensioal tensor product Gauss-Legendre quadrature rule (4.8) to approximate the integrals from (4.7), we obtain the approximation

$$\tilde{\boldsymbol{A}}_{\boldsymbol{\lambda},\boldsymbol{\lambda}'} = \sum_{(j,\boldsymbol{k})\in\mathcal{I}}\sum_{(j',\boldsymbol{k}')\in\mathcal{I}'}\sum_{l=1}^{n^2}\sum_{l'=1}^{n'^2}\boldsymbol{\omega}_l\,\boldsymbol{\omega}_{l'}\,k_{i,i'}(\boldsymbol{\xi}_l,\boldsymbol{\xi}_{l'})\,w_{\boldsymbol{k}',\iota'}\,w_{\boldsymbol{k},\iota} \tag{4.10}$$

to $\boldsymbol{A}_{\boldsymbol{\lambda},\boldsymbol{\lambda}'}$. Hence, the use of numerical quadrature introduces a perturbed matrix entries $\tilde{\boldsymbol{A}}_{\boldsymbol{\lambda},\boldsymbol{\lambda}'}$. Despite of this error, we still want a scheme which guarantees optimal convergence and complexity.

### 4.3.7 Choice of the Degree of Quadrature

We will now go into detail on the accuracy which has to be realised by the quadrature in order to produce an over-all error which is consistent with the theory from Subsection 3.3.3. Suppose that we have given an $H^s$-normalised wavelet basis. Recall the representation for the approximation to the matrix-vector product, i.e.

$$\left\|\boldsymbol{A}\boldsymbol{v} - \sum_{j=1}^{J}\boldsymbol{A}_j\boldsymbol{v}|_{\Delta_j}\right\|_{\ell^2(\mathcal{J}_\star)} \lesssim \epsilon.$$

Here, the layers $\Delta_j$ are formed by taking the difference of the two subsequent trees $\mathcal{J}_{j-1}\backslash\mathcal{J}_j$, where $\mathcal{J}_J \subset \mathcal{J}_{J-1} \subset \cdots \subset \mathcal{J}_0 = \mathcal{J}$ and $J$ is obtained by (3.22). For computing the approximation $\boldsymbol{w}_{\mathcal{J}} = \sum_{j=1}^{J}\boldsymbol{A}_j\boldsymbol{v}|_{\Delta_j}$ to the infinite matrix-vector product $\boldsymbol{A}\boldsymbol{v}$, we use tensor product Gauss-Legendre quadrature rules. Hence, an error is introduced. To that end, let us introduce the notation $\boldsymbol{A}_j^{\Delta_j} = (\boldsymbol{A}_j)_{\boldsymbol{\lambda}\in\mathcal{J},\boldsymbol{\lambda}'\in\Delta_j}$ and $\tilde{\boldsymbol{A}}_j^{\Delta_j}$ for the approximation of $\boldsymbol{A}_j^{\Delta_j}$ by quadrature. Furthermore, $\tilde{\boldsymbol{w}}_{\mathcal{J}} = \sum_{j=1}^{J}\tilde{\boldsymbol{A}}_j^{\Delta_j}\,\boldsymbol{v}|_{\Delta_j}$ shall denote the approximation to $\boldsymbol{w}_{\mathcal{J}}$. This approximation shall fulfil the estimate

$$\|\boldsymbol{A}\boldsymbol{v} - \tilde{\boldsymbol{w}}_{\mathcal{J}}\|_{\ell^2(\mathcal{J}_\star)} \lesssim \epsilon. \tag{4.11}$$

If the error, which is introduced by approximating the blocks $(\tilde{\boldsymbol{A}}_j^{\Delta_j})_{\boldsymbol{\lambda},\boldsymbol{\lambda}'}$, stays proportional to $2^{-j\bar{s}}$ for all $j \le J$, then obviously estimate (4.11) holds, see e.g. [27,28,54]. The error

which is allowed per entry has been identified in [28]:

**Proposition 35.** If the used quadrature realises the accuracy

$$\left|(\boldsymbol{A}_j^{\Delta_j})_{\boldsymbol{\lambda},\boldsymbol{\lambda}'} - (\tilde{\boldsymbol{A}}_j^{\Delta_j})_{\boldsymbol{\lambda},\boldsymbol{\lambda}'}\right| \leq \frac{2^{-j\bar{s}}}{\sqrt{2^j \min\left(2^j, \#\Delta_j/\alpha_j\right)}} =: \epsilon_{j,j'}, \tag{4.12}$$

then the error, which is introduced by the approximation of the blocks $(\tilde{\boldsymbol{A}}_j^{\Delta_j})_{\boldsymbol{\lambda},\boldsymbol{\lambda}'}$, stays proportional to $2^{-j\bar{s}}$ for all $j \leq J$.

For the computation of the matrix entries, we use the four-dimensional tensor product Gauss-Legendre quadrature rule to approximate the element-element interactions. If we want that the approximation realises the desired accuracy, the elements $\Gamma_{i,j,\boldsymbol{k}}$ and $\Gamma_{i',j',\boldsymbol{k}'}$ have to fulfil the criterion

$$\operatorname{dist}(\Gamma_{i,j,\boldsymbol{k}}, \Gamma_{i',j',\boldsymbol{k}'}) \geq 2^{-\min\{j,j'\}}. \tag{4.13}$$

Whenever this criterion is fulfilled, we use the estimate (4.9) to compute the degree of quadrature, which is necessary in order to ensure the desired precision from (4.12). Thereby, we need to take into account that we use an $H^s$-normalised wavelet basis. In view of (4.9), by setting the degrees of quadrature $n$ and $n'$ to

$$n = \left\lceil \frac{z}{\log_2(\operatorname{dist}(\Gamma_{i,j,\boldsymbol{k}}, \Gamma_{i',j',\boldsymbol{k}'})) + j + 2 + \log_2(r)} \right\rceil,$$
$$n' = \left\lceil \frac{z}{\log_2(\operatorname{dist}(\Gamma_{i,j,\boldsymbol{k}}, \Gamma_{i',j',\boldsymbol{k}'})) + j' + 2 + \log_2(r)} \right\rceil,$$

with

$$z = \left\lceil -\frac{1}{2}\left(\log_2(\epsilon_{j,j'}) + (j+j')(1+s) + (2+2s)\log_2(\operatorname{dist}(\Gamma_{i,j,\boldsymbol{k}}, \Gamma_{i',j',\boldsymbol{k}'}))\right) \right\rceil,$$

the desired precision is met. In our implementation, the choice $r = 1$ turns out to be sufficient.

In the case that the criterion (4.13) is not fulfilled, the quadrature error does not tend to zero when increasing $n$ and $n'$. If $\Gamma_{i,j,\boldsymbol{k}} \cap \Gamma_{i',j',\boldsymbol{k}'} \neq \emptyset$, i.e. if either the elements are the same or if they have a common edge or a common vertex, the integrand is singular. In these cases, we use the so-called Duffy trick to transform the singular integrands into non-singular ones. For the details, see e.g. [35,54,81].

### 4.3.8 Computation of the Matrix Entries

Certain elements, for which the criterion (4.13) does not hold, may have to be subdivided further for calculating the desired integral by either using or computing the integrals with respect to their children. One example would be the integration of a very coarse element

$\Gamma_{i,j,\boldsymbol{k}}$ which is in the neighbourhood of a very small element $\Gamma_{i',j',\boldsymbol{k}'}$ such that (4.13) is not satisfied. In such a case, we use the integrals which have been computed for the four children of $\Gamma_{i,j,\boldsymbol{k}}$ to obtain the integral of the parent. On one hand, this means that the integral for the parent does not have to be calculated from scratch, but more importantly, these integrals can afterwards also be used for the calculation of other element-element interactions featuring this element or one of its ancestors. This procedure we refer to as recycling. Suppose the element $\Gamma_{i,j,\boldsymbol{k}}$ has four children $\Gamma_{i,j+1,\boldsymbol{k}_l}$ where $\boldsymbol{k}_0 = (k_1, k_2)$, $\boldsymbol{k}_1 = (k_1 + 2^{-(j+1)}, k_2)$, $\boldsymbol{k}_2 = (k_1 + 2^{-(j+1)}, k_2 + 2^{-(j+1)})$ and $\boldsymbol{k}_3 = (k_1, k_2 + 2^{-(j+1)})$, and their element-element interactions $v_{(i,j+1,\boldsymbol{k}_l),(i',j',\boldsymbol{k}')}$ with the element $\Gamma_{i',j',\boldsymbol{k}'}$ are known. Then, the element-element interaction $v_{(i,j,\boldsymbol{k}),(i',j',\boldsymbol{k}')}$ for the interaction of the parent $\Gamma_{i,j,\boldsymbol{k}}$ with $\Gamma_{i',j',\boldsymbol{k}'}$ can be obtained by

$$\begin{aligned} v_{(i,j,\boldsymbol{k}),(i',j',\boldsymbol{k}')} = \frac{1}{2}\Big( & v_{(i,j+1,\boldsymbol{k}_0),(i',j',\boldsymbol{k}')} + v_{(i,j+1,\boldsymbol{k}_1),(i',j',\boldsymbol{k}')} + \\ & v_{(i,j+1,\boldsymbol{k}_2),(i',j',\boldsymbol{k}')} + v_{(i,j+1,\boldsymbol{k}_3),(i',j',\boldsymbol{k}')}\Big). \end{aligned} \tag{4.14}$$

There appear two challenges in the implementation. One challenge concerns the precision and the other challenge concerns the memory requirement. First, the accuracies which are needed for different element-element interactions vary. Second, storing the element-element interactions for recycling for all elements in the tree is not efficient, since it would need too much memory.

We solve both problems by using a queueing system for the involved elements. Recall that, for compression, we used a sorted tree with respect to the wavelet layer. Between compression and assembling, the sorting will certainly not change, since the indexing would then no longer be consistent. We use the sorted wavelet list to determine the layer of an element

```
e → layer = max{ w → layer  for all  w ∈ e → wavelet }.
```

As soon as we know the number of elements per layer, we arrange them in a queue, starting with the largest layer. For elements with a larger layer, we need to assign a larger degree of quadrature in order to guarantee that quadrature is performed within the desired accuracy. As soon as the queue has been created, we start the assembly routine which iterates over the elements in the queue. Since the routine for this iteration is long and requires a lot of explanation, we present it in two pieces, part one of which forms Algorithm 11.

There shall be given the $i$-th element in the queue, which is contained in the support of several different wavelets (the number of wavelets which have this element in their support is given in the variable `e → waveNumber`). First, the weights associated with this element in each of those wavelets is stored in the array `double *weights`, which has been allocated before. Second, in preparation for the part of the algorithm, which is outlined in Algorithm 12, a second array `ansatzWave` is allocated.

The do-while loop, which is outlined in Algorithm 12, consists of an iteration through all

**Algorithm 11:** First part of the iteration for computing element-element interactions.

Given: $i$-th element `e` of the queue, i.e. `e = queue[`$i$`]`;
Allocate: `ansatzWave = zeros(e` $\rightarrow$ `waveNumber)`;
Allocate: `weights = zeros(e` $\rightarrow$ `waveNumber)`;
**for** *All wavelets* `w` *in* `e` $\rightarrow$ `wavelet[`$j$`]` **do**
  Find $k$ such that `e` $\rightarrow$ `wavelet[`$j$`]` $\rightarrow$ `support[`$k$`] = e`;
  Set: `weights[`$j$`] = waveWeights[kind(w)][w` $\rightarrow$ `protoType][`$k$`]`;
**end**

wavelets which interact with one of the wavelets having the element `e` in their support. The first step is to find the minimal index `ind` for which we have to compute an element-element interaction. With this index, we automatically have the correct wavelet at hand, i.e. the wavelet `testWave` = `sortVec[ind]`. Next, we compute the interaction between the element `e` and each element in the support of `testWave`. The value `c` which is obtained by the routine INTERACTION (this routine is explained in Algorithm 13 later), is then multiplied with the weight associated with the element in the wavelet's support to obtain the element-element interaction `result`.

**Algorithm 12:** Second part of the iteration for computing element-element interactions.

**do**
  **for** $j = 0$ **to** `e` $\rightarrow$ `waveNumber` **do**
    Find: min ind = `A` $\rightarrow$ `index[e` $\rightarrow$ `wavelet[`$j$`]` $\rightarrow$ `index][ansatzWave[`$j$`]]`;
  **end**
  **if** `ind = A` $\rightarrow$ `noRows` **then** break;
  Set: `testWave` = `sortVec[ind]`;
  Set: `result` = 0;
  **for** $j = 0$ **to** `testWave` $\rightarrow$ `supportSize` **do**
    c = INTERACTION (`e, testWave` $\rightarrow$ `support[`$j$`]`);
    Set: `result +=` c$\cdot$ `waveWeights[kind(testWave)][testWave` $\rightarrow$ `protoType][`$j$`]`;
  **end**
  **for** $j = 0$ **to** `e` $\rightarrow$ `waveNumber` **do**
    **if** `(A` $\rightarrow$ `index[e` $\rightarrow$ `wavelet[`$j$`]` $\rightarrow$ `index][ansatzWave[`$j$`]] = testWave)`
    **then**
      Add `result` $\cdot$ `weights[`$j$`]` to correct position in the matrix;
      Increase: `++ansatzWave[`$j$`]`;
    **end**
  **end**
**while** *no* break;
Free all variables which are not needed any more;

Finally, the desired element-element interaction, more precisely `result` $\cdot$ `weights[j]`, is added to the correct position in the matrix. In this step, we also increase the index of

the ansatz wavelet `ansatzWave`, such that the next larger index `ind` will be found at the beginning of the next iteration of this do-while loop. This procedure continues until the minimal index, which can be found in `A` → `index`, is the dimension of the matrix, in which case the iteration is terminated. The program then proceeds to the next element in the queue.

**Algorithm 13:** double = INTERACTION(`e1, e2`)

```
Find and set: first, second (element with smaller/larger index);
if j = searchIntegral (first, second) ≠ -1 then
    return first → integrals.value[j];
end
if e1 → level = e2 → level then
    Calc: dist(e1, e2);
    Calc: [g1, g2] = degOfQuad (e1 → level, e2 → level, dist(e1, e2), prec);
    Calc: case;
    case 1: do result = NORMAL;
    case 2: do result = PATCH;
    case 3: do result = EDGE;
    case 4: do result = VERTEX;
else // e1 → level != e2 → level
    Find and set: high, low (element with higher/lower level);
    if dist(e1, e2) is large enough then
        result = NORMAL;
    else
        if low has no children then refineElement (low);
        for i = 0 to 3 do
            result += 0.5 · INTERACTION (hi, low → child[i]);
        end
    end
end
Set: setIntegral(first, second, int);
return int;
```

Inside this procedure, the routine INTERACTION is called, which is where the quadrature is performed.

INTERACTION has two elements `e_1` and `e_2` as input. Each element has assigned an index that coincides with its position in the queue. In doing so, we create a hierarchy on the elements, which we can use for the recycling of the integrals. As we always calculate the integral between two elements, it is not trivial where to store the element-element interaction and, for memory reasons, it is no valid option to just store it for both. We decided to store the element-element interaction in the structure associated with the element having the smaller index, i.e. the element which is more to the front in the queue. This is the

natural choice, considering the above assembly routine running through the queue.

As the routine INTERACTION is called, it first checks whether the desired element-element interaction has already been calculated in a previous call and has been stored. If this is the case, the element-element interaction is returned and the routine terminates. For what follows, we distinguish between two cases. Either the elements `e1` and `e2` have the same level, in which case we decide which of the following four quadrature routines shall be used to perform the quadrature, or the level differs. For elements with the same level, we distinguish between four cases: NORMAL, PATCH, EDGE and VERTEX. Each of these routines is associated with how the two elements are aligned. If they share a vertex, an edge, or if they are the same, the kernel contains the singularity and the Duffy trick, see e.g. [35, 54, 81], must be applied in order to transform the singular integral into a non-singular integral. In the latter case, i.e. if the level differs, the distance between the elements may be large enough such that (4.13) applies and direct quadrature can be used. If the distance is too small, i.e. (4.13) is violated, then the larger element is refined and the routine INTERACTION is called for all children in the further recursion. As soon as an element-element interaction is computed, it will be stored with the first element by the routine SETINTEGRAL.

After all element-element interactions are calculated and added to the system matrix for one element in the queue, it will never appear again in the assembly routine. This is why we can clear all variables in the according element associated with the integration routine.

By calling the routine for all elements in the order in which they are aligned in the queue, we may automatically compute values for descendants with a higher accuracy than necessary. Recall that the matrix-vector product can schematically be written as

$$\boldsymbol{A}\boldsymbol{v} \approx \overset{\sim 2^{-J\bar{s}}}{\downarrow}\ \overset{\sim 2^{-(J-1)\bar{s}}}{\downarrow}\ \overset{\sim 1}{\downarrow} \left[\begin{array}{c|c|c|c} \boldsymbol{A}_J & \boldsymbol{A}_{J-1} & \cdots & \boldsymbol{A}_0 \end{array}\right] \left[\begin{array}{c} \boldsymbol{v}|_{\Delta_J} \\ \hline \boldsymbol{v}|_{\Delta_{J-1}} \\ \hline \vdots \\ \hline \boldsymbol{v}|_{\Delta_0} \end{array}\right] \begin{array}{l} \leftarrow \sim \epsilon\, 2^{J\bar{s}} \\ \leftarrow \sim \epsilon\, 2^{(J-1)\bar{s}} \\ \\ \leftarrow \sim \epsilon \end{array} . \tag{4.15}$$

For each $j$, we denote by $\boldsymbol{A}_j$ the compressed sub-matrices as described in Subsection 4.3.3, where $j = \{0, \cdots, J\}$ and $J$ given by equation (3.22)) we denote the part of the matrix which contains entries for the particular layer $J$. To be precise, in this block, all non-negligible matrix entries

$$(\boldsymbol{A}_j)_{\boldsymbol{\lambda},\boldsymbol{\lambda}'} \quad \text{for } \boldsymbol{\lambda} \in \mathcal{J} \text{ and } \boldsymbol{\lambda}' \in \Delta_j$$

are found. The visualisation of the approximation for the matrix-vector product $\boldsymbol{A}\boldsymbol{v}$ in (4.15) is just the basic idea, but not what is actually done in practice. Inside the blocks $\boldsymbol{A}_j$, for $j = 0, \ldots, J$, we can further consider the portion of each layer, leading to the blocks $\boldsymbol{A}_{j,j'}$ given by

$$\left(\boldsymbol{A}_{j,j'}\right)_{\boldsymbol{\lambda},\boldsymbol{\lambda}'} \quad \text{for } \boldsymbol{\lambda} = \Delta_j \text{ and } \boldsymbol{\lambda}' \in \Delta_{j'}$$

with $j, j' \in \{0, \dots, J\}$. This pattern is visualised in the upper left part of (4.16):

$$\begin{array}{|c|c|c|c|} \hline \boldsymbol{A}_{J,J} & \boldsymbol{A}_{J,J-1} & \cdots & \boldsymbol{A}_{J,0} \\ \hline \boldsymbol{A}_{J-1,J} & \boldsymbol{A}_{J-1,J-1} & \cdots & \boldsymbol{A}_{J-1,0} \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline \boldsymbol{A}_{0,J} & \boldsymbol{A}_{0,J-1} & \cdots & \boldsymbol{A}_{0,0} \\ \hline \end{array} \rightsquigarrow \begin{array}{|c|c|c|c|} \hline \boldsymbol{A}_{J,J} & \boldsymbol{A}^t_{J-1,J} & \cdots & \boldsymbol{A}^t_{0,J} \\ \hline \boldsymbol{A}_{J-1,J} & \boldsymbol{A}_{J-1,J-1} & \cdots & \boldsymbol{A}_{J-1,0} \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline \boldsymbol{A}_{0,J} & \boldsymbol{A}_{0,J-1} & \cdots & \boldsymbol{A}_{0,0} \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|c|} \hline \boldsymbol{A}_{J,J} & \boldsymbol{A}^t_{J-1,J} & \cdots & \boldsymbol{A}^t_{0,J} \\ \hline \boldsymbol{A}_{J-1,J} & \boldsymbol{A}_{J-1,J-1} & \cdots & \boldsymbol{A}_{J-1,0} \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline \boldsymbol{A}_{0,J} & \boldsymbol{A}_{0,J-1} & \cdots & \boldsymbol{A}_{0,0} \\ \hline \end{array} \rightsquigarrow \begin{array}{|c|c|c|c|} \hline \boldsymbol{A}_{J,J} & \boldsymbol{A}^t_{J-1,J} & \cdots & \boldsymbol{A}^t_{0,J} \\ \hline \boldsymbol{A}_{J-1,J} & \boldsymbol{A}^t_{J-1,J-1} & \cdots & \boldsymbol{A}^t_{0,J-1} \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline \boldsymbol{A}_{0,J} & \boldsymbol{A}_{0,J-1} & \cdots & \boldsymbol{A}_{0,0} \\ \hline \end{array} \tag{4.16}$$

The entries in the first block $(\boldsymbol{A}_j)_{\boldsymbol{\lambda},\boldsymbol{\lambda}'}$ with $\boldsymbol{\lambda} \in \mathcal{J}$ and $\boldsymbol{\lambda}' \in \Delta_j$ have to be calculated with the highest precision $2^{-J\bar{s}}$, the entries in the second block $(\boldsymbol{A}_j)_{\boldsymbol{\lambda},\boldsymbol{\lambda}'}$ with $\boldsymbol{\lambda} \in \mathcal{J}$ and $\boldsymbol{\lambda}' \in \Delta_{j-1}$ have to be calculated with the second highest precision $2^{-(J-1)\bar{s}}$, and so on. What is done in practice is the following. First, whenever an entry $\langle \mathcal{A}\psi_{\boldsymbol{\lambda}'}, \psi_{\boldsymbol{\lambda}} \rangle$ is relevant according to the compression pattern (4.4), the entry $\langle \mathcal{A}\psi_{\boldsymbol{\lambda}}, \psi_{\boldsymbol{\lambda}'} \rangle$ is relevant too, leading to a symmetric compression pattern. Second, if entries $(\boldsymbol{A}_j)_{\boldsymbol{\lambda},\boldsymbol{\lambda}'}$ are calculated with $|\boldsymbol{\lambda}'| \geq |\boldsymbol{\lambda}|$, demanding for high precision, we can cheaply obtain the mirrored entries simultaneously. Thus, the entries above the diagonal are calculated with an even higher precision than necessary, where the cost is at most doubled. This idea is illustrated in (4.16).

### 4.3.9 Rhs

With the routine apply at hand, we have an important ingredient for calculating the residual. We need still one more ingredient, namely a routine for calculating the right-hand side of the system of linear equations. As mentioned in Subsection 3.3.4, this will be done in the routine $\boldsymbol{f}_{\mathcal{J}} = \text{Rhs}[\epsilon]$. For a desired target accuracy $\epsilon$, it returns the finitely supported approximation $\boldsymbol{f}_{\mathcal{J}}$ to the right-hand side $\boldsymbol{f}$ such that

$$\|\boldsymbol{f}_{\mathcal{J}} - \boldsymbol{f}\|_{\ell^2(\mathcal{J}_\star)} \leq \epsilon.$$

This is to be calculated in linear complexity $\mathcal{O}(N)$, with $N$ denoting the degrees of freedom. In order to realize this, we need a-priori information on the function $\boldsymbol{f}$.

For the evaluation of the right-hand side $f\colon \mathbb{R}^3 \to \mathbb{R}$, we have to approximate the scalar product of the according function with a wavelet. Similar (but easier) as for the matrix assembly, we split the wavelet into its fixed set of elements in its support. Thus, we need to calculate a sum of integrals for each element. We use again the element-wise representation (4.2) to get

$$\boldsymbol{f}_{\boldsymbol{\lambda}} = \langle f, \psi_{\boldsymbol{\lambda}} \rangle = \int_{\square} f_i(\boldsymbol{s})\, \psi_{\boldsymbol{\lambda}}(\boldsymbol{\gamma}_i(\boldsymbol{s}))\, \mathrm{d}\boldsymbol{s} = \sum_{(j,\boldsymbol{k}) \in \mathcal{I}} 2^j\, \omega_{\boldsymbol{k},\iota} \int_{\square_{j,\boldsymbol{k}}} f_i(\boldsymbol{s})\, \mathrm{d}\boldsymbol{s}.$$

Here, we abbreviated $f_i(\boldsymbol{s}) \coloneqq f(\boldsymbol{\gamma}_i(\boldsymbol{s}))$, where we used the usual transformation to the unit square again. Analogously to Subsection 4.3.6, where we used a four-dimensional tensor product Gauss-Legendre quadrature, we use a two-dimensional $n$-point tensor product Gauss-Legendre quadrature

$$Q^{(n)}_{(\square_{j,\boldsymbol{k}})}[f] \coloneqq \sum_{l=1}^{n^2} \boldsymbol{\omega}_l \, f(\boldsymbol{\xi}_l),$$

to approximate a two-dimensional integral of the form $I_{(\square_{j,\boldsymbol{k}})}[f] \coloneqq 2^j \int_{\square_{j,\boldsymbol{k}}} f(\boldsymbol{s}) \mathrm{d}\boldsymbol{s}$. With the help of a suitable quadrature rule, we compute the approximation to the right-hand side according to

$$\tilde{\boldsymbol{f}}_{\boldsymbol{\lambda}} = \sum_{(j,\boldsymbol{k})\in\mathcal{I}} \sum_{l=1}^{n^2} \boldsymbol{\omega}_l f_i(\boldsymbol{\xi}_l) \, w_{\boldsymbol{k},\iota}. \tag{4.17}$$

The implementation in this case is not as tricky as in the assembly routine. For each element, we store the approximated integral of the function $f$ in the variable `double load`. This variable is not set from the beginning whenever an element is created, but it is computed several times during the implementation and may also change its value. As the adaptive algorithm advances, we increase the precision to compute these integrals. Thus, in the growing routine, we call the routine UPDATELOAD to replace the values by a more accurate approximation, before we call the routine RHS for computing the right-hand side. The routine UPDATELOAD is outlined in Algorithm 14 below.

**Algorithm 14:** UPDATELOAD(`e`, $\epsilon$)

Input: Element e, accuracy $\epsilon$

```
if e → child[0] ≠ NULL then
    for all children do
        updateLoad(e → child[i], ε);
    end
    e → load = 0.5 * (e → child[0] → load + e → child[1] → load +
                      e → child[2] → load + e → child[3] → load);
else
    makeQuadrature(e, ε);
end
```

This function is called recursively for each element which is currently in the tree. The routine MAKEQUADRATURE, that performs the calculation in (4.17), is called only for the leafs of the tree. All remaining integrals are computed by combining the four integrals of the element's children. After each element has this integral stored, we can call the routine RHS. Therein, we combine these integrals with the correct wavelet weights, in order to calculate the approximation to the right-hand side for each wavelet.

### 4.3.10 COARSE

The last subsection in this chapter is dedicated to the coarsening routine. In the routine GROW, we are expanding the (tree structured) index set $\mathcal{J}$ from the previous iteration to the larger set $\mathcal{J} \subset \mathcal{J}'$ in such a way that the error is reduced by a constant factor. To control the complexity, we have to coarsen the index set $\mathcal{J}'$ such that $\|\boldsymbol{r}_{\mathcal{J}''}\| \leq \theta \|\boldsymbol{r}_{\mathcal{J}'}\|$ for a $\theta \in (0,1)$ which is small enough. Note that $\mathcal{J}''$ is still a refined version of $\mathcal{J}$, i.e. $\mathcal{J} \subset \mathcal{J}'' \subset \mathcal{J}'$.

Suppose that we have a sorted wavelet tree at hand. Which wavelets we are going to deactivate in the coarsening routine is then decided by thresholding. As mentioned earlier, there are different norms we use for the sorting. Before coarsening the residual, we used the routines APPLY and RHS to compute it. As soon as this step has been performed, we will sort the residual vector with respect to the norm $\|\cdot\|_{-q}$ from equation (4.3) by using the routine outlined in Subsection 4.3.2.

For calling the coarsening routine, we will additionally compute two quantities `outerNorm` and `innerNorm`. To obtain them, we will distinguish between permanently added wavelets, having their activity set to 1, and wavelets currently in a test-phase, with activity 2. The inner norm is computed by summing up all coefficients of wavelets with activity 1 in (4.3). The outer norm consists of the coefficients of wavelets with activity 2. The threshold parameter, which is finally included in the routine, is obtained by multiplying the outer norm with the threshold constant. The routine for coarsening is outlined in Algorithm 15 below.

**Algorithm 15:** void COARSERES(thresh · `outerNorm`)

Set: err $= i = 0$
Set: eps = thresh · `outerNorm`
**while** eps · eps + err < `Norm` **do**
    Add: err `+= (wavelet[`$i$`]` $\rightarrow$ `res)`$^2$ / `wavelet[`$i$`]` $\rightarrow$ `diag[`$i$`];`
    Set: `++`$i$;
**end**
**for** $j = i$ **to** end **do**
    Set: `wavelet[`$j$`]` $\rightarrow$ `activity = 0;`
**end**

**Remark 36.** One additional issue, which could be discussed, is the deactivation of the wavelet. It would be a possibility to remove the unnecessary wavelets from the tree instead of setting its activity to 0, having the advantage that no unnecessary memory is occupied. However, if the wavelet is to be added again in the next prediction step, it has to be created again.

With this remark, we close this chapter on the implementation of the adaptive wavelet method and proceed to Chapter 5, where we are going to present various numerical results.

# 5

# Numerical Experiments

In this chapter, we will present our numerical results for the adaptive wavelet method, which has been discussed in theory in the previous Chapters 3 and 4. We have developed an adaptive code for solving the interior Laplace equation by using the single layer operator or by using the double layer operator. Also, we have developed an adaptive code for solving the exterior Helmholtz equation for wavenumbers $\kappa \geq 1$ by the Brakhage-Werner formulation. Furthermore, we present results for solving the scattering problem involving a sound-soft scatterer. All developed codes work for various geometries and right-hand sides. The subsequent experiments have all been performed on a single processor of a computing server with 24 Intel(R) Xeon(R) E5-2643 CPUs with a clock rate of 3.40GHz and a main memory of 256GB. For the visualisation of the results, the program Paraview has been used.

This chapter is structured as follows: The first section will be a parameter study. Therein, we focus on one representative example and investigate how the change of certain parameters influences the estimated residual or other quantities of interest like the refinement and coarsening behaviour of the code. Based on these results, we will fix one setting which is then used for all following experiments. In the second section, we present our first results for solving the interior Laplace problem by using an indirect formulation which involves the single layer potential, leading to a Fredholm integral equation of the first kind for the single layer operator. We will consider various geometries as well as different smooth right-hand sides. The third section is concerned with presenting some results for the Laplace equation, which has been solved by an indirect formulation which involves the double layer potential, leading to a Fredholm integral equation of the second kind involving the double layer operator. In the course of our experiments we will notice, in contrast to the boundary integral equation for the single layer operator, that adaptivity does not pay off for

smooth right-hand sides. This is still the case, even though we also consider geometries featuring edges and vertices, see [24]. Motivated by this, we subsequently choose a singular right-hand side of the form $f(r) = 1/r^{\alpha}$, where the expected rate of convergence for adaptive and uniform refinement depends on the parameter $\alpha$. For uniform refinement, we expect to obtain only half the rate as for an adaptive approach, provided that $\alpha$ is sufficiently small. We will choose various $\alpha$ and observe the predicted behaviour, which is in accordance with the theory proposed in [24]. The fourth section is concerned with solving the exterior Helmholtz equation for different wavenumbers $\kappa \geq 1$. The observation that adaptivity is not necessary for solving the Laplace equation by means of the double layer potential if a smooth right-hand side is chosen, can also be made for the exterior Helmholtz equation. This behaviour motivates the choice of discontinuous Dirichlet data for the first numerical example in Section 5.4. In this experiment, we consider only the case of a fixed wavenumber $\kappa = 1$. In the forthcoming numerical examples, we consider then scattering problems for various $\kappa \geq 1$ at a drilled cube. Now, since we use the direct formulation (1.26), adaptivity pays off.

## 5.1 Parameter Study

Before we present our numerical results, we will first study some parameters chosen for our experiments. In particular, this section is concerned with the increase of the accuracy during GROW and before SOLVE.

During one loop of GROW, we increase the accuracy right before the matrix compression (see Figure 4.9). This means that we shift each layer $j$ by a fixed constant $l$:

$$\left[\begin{array}{c|c|c|c} \boldsymbol{A}_{J,J} & \boldsymbol{A}_{J,J-1} & \cdots & \boldsymbol{A}_{J,0} \\ \hline \boldsymbol{A}_{J-1,J} & \boldsymbol{A}_{J-1,J-1} & \cdots & \boldsymbol{A}_{J-1,0} \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline \boldsymbol{A}_{0,J} & \boldsymbol{A}_{0,J-1} & \cdots & \boldsymbol{A}_{0,0} \end{array}\right] \rightsquigarrow \left[\begin{array}{c|c|c|c} \boldsymbol{A}_{J+l,J+l} & \boldsymbol{A}_{J+l,J-1+l} & \cdots & \boldsymbol{A}_{J+l,l} \\ \hline \boldsymbol{A}_{J-1+l,J+l} & \boldsymbol{A}_{J-1+l,J-1+l} & \cdots & \boldsymbol{A}_{J-1+l,l} \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline \boldsymbol{A}_{l,J+l} & \boldsymbol{A}_{l,J-1+l} & \cdots & \boldsymbol{A}_{l,l} \end{array}\right].$$

Obviously, if we choose a higher accuracy for the assembling inside the growing routine, we end up with a more accurate version of the matrix, which results in a more accurate computation of the residual. Keeping in mind that quadrature is the most time consuming part of the implementation (and accuracy controlling a crucial part of the chosen degree of quadrature), we still do not want to prescribe a higher accuracy than necessary.

After the inner loop of growing the wavelet tree is complete, we will solve the system of linear equations for the newly enlarged index set (see Figure 4.9). As already mentioned, we are using an iterative solver, namely CG [62] if the matrix is symmetric and positive definit and GMRES [80] if not. Thus, for SOLVE, we need to assemble the matrix once more. Here, we can decide to increase the accuracy again, which then results in a more accurate version of the solution. For both mentioned situations, the accuracy used for

matrix assembly has to be chosen high enough in order to ensure a sufficiently accurate residual or solution, such that the convergence of the estimated residual and the desired approximation error of the density is guaranteed.

We have two options: The accuracy may be chosen tailor-made, i.e. differently, for solving and for calculating the residual, as the requirements on accuracy may vary. However, since matrix assembly is the most time consuming part of the implementation, it might be more effective to choose a higher accuracy for growing only and use a subset of this matrix for solving the system of linear equation associated with the coarsened index set. Both of these options have pros and cons.

1. Choosing the accuracy differently for solving and calculating the residual gives us more control on the individuality of the situation. We can choose different accuracies that are especially adapted for either solving the system of linear equations or for calculating the residual. This means that we do not have to choose the accuracy unnecessarily high for growing in order to ensure a sufficiently accurate matrix for solving or vice versa. However, in this case, we need to assemble the matrix at least twice, which might be unnecessarily time consuming.

2. If we choose the accuracy in the growing routine only and solve the system of linear equations only on a subset of the underlying matrix, we are not able to control the accuracy independently for the different situations. It is possible that we calculate the residual with a too high accuracy (or just the right one) but solve with insufficient accuracy. The opposite could be the case, too: In order for the complete scheme to converge, we may have to choose a ridiculously high accuracy in the growing routine, which would not be necessary for growing alone, since we use the same matrix for solving where we need a higher accuracy. This will get especially expensive, if more than one growing step is needed before we proceed to solving the system of linear equations. Nevertheless, if the growing step is performed once, we only need to assemble the matrix once and we save a lot of time. We do not expect to gain a factor two in computation time, since the matrix which we have to assemble for solving will be smaller than the matrix for calculating the residual, due to the coarsening after GROW.

This ambivalence led to the following parameter study. Since it is a priori not clear which option is the better one, meaning which of the aforementioned statements is predominant, we perform some experiments to find that out.

As a representative situation, we consider the exterior Helmholtz problem with Dirichlet boundary conditions

$$\begin{aligned} \Delta u + \kappa^2 u &= 0 && \text{in } \Omega^c := \mathbb{R} \setminus \bar{\Omega}, \\ u &= f && \text{on } \Gamma, \\ \partial u/\partial r - i\kappa u &= \mathcal{O}(1/r^2) && \text{for } r = \|\boldsymbol{x}\| \to \infty, \end{aligned} \tag{5.1}$$

for the Fichera vertex ($\Omega = (0,1)^3 \setminus (0,0.5]^3$). As Dirichlet data in (5.1), we choose the restriction $f = u|_\Gamma$ of the complex function $u(\boldsymbol{x}) = \Re(\boldsymbol{x}) + \mathrm{i}\Im(\boldsymbol{x})$ with

$$\Re(\boldsymbol{x}) = \frac{\cos(\kappa\|\boldsymbol{x}-\boldsymbol{a}\|)}{\|\boldsymbol{x}-\boldsymbol{a}\|}, \quad \Im(\boldsymbol{x}) = \frac{\sin(\kappa\|\boldsymbol{x}-\boldsymbol{a}\|)}{\|\boldsymbol{x}-\boldsymbol{a}\|}. \tag{5.2}$$

The wavenumber $\kappa$ is set to $\kappa = 1$ and the parameter $\boldsymbol{a}$ is set to $\boldsymbol{a} = (0.75, 0.75, 0.75)$, which is located inside Fichera's vertex. As one readily verifies, the function $u = u|_{\Omega^c}$ solves the Helmholtz problem (5.1).

We tested different combinations of accuracy shifts $l_1$ inside the growing routine and $l_2$ before solving the system of linear equations. We supervised the convergence of the following two quantities. The first quantity is the estimated norm of the residual **res** in the $\ell^2$-norm, denoted by $\|\mathbf{res}\|$. The second quantity is the maximal error of the potential, denoted by $\|\mathbf{u} - \mathbf{u}_{N_{\mathrm{dof}}}\|_\infty$. In view of the definition for the evaluation of the potential from Chapter 1, we have the estimate

$$\begin{aligned} |(\mathbf{u} - \mathbf{u}_{N_{\mathrm{dof}}})(\boldsymbol{x}_i)| &= \left| \int_\Gamma k(\boldsymbol{x}_i, \boldsymbol{y}) \, (\rho - \rho_{N_{\mathrm{dof}}})(\boldsymbol{y}) \, \mathrm{d}\sigma_{\boldsymbol{y}} \right| \\ &\leq \|k(\boldsymbol{x}_i, \cdot)\|_{H^{-s}(\Gamma)} \, \|\rho - \rho_{N_{\mathrm{dof}}}\|_{H^s(\Gamma)} \end{aligned} \tag{5.3}$$

for each point $\boldsymbol{x}_i \in \Omega^c$. Hereby, $\rho$ denotes the exact solution of the associated boundary integral equation and $\rho_{N_{\mathrm{dof}}}$ denotes the numerical solution. Taking the maximum norm is then a natural choice. The potential is evaluated after the involved tree has been coarsened and the involved system of linear equations has been solved numerically. The norm of the estimated residual is computed on the enlarged index set after the prediction inside GROW. By $N_{\mathrm{dof}}$ we denote the degrees of freedom after coarsening the tree and by $N_{\mathbf{res}}$ we denote the degrees of freedom after prediction. The vector $\mathbf{u} = [u(\boldsymbol{x}_i)]_i$ hereby stands for the evaluation of the function $u$ in several points $\boldsymbol{x}_i$ inside the domain $\Omega$ for interior problems, or inside the domain $\Omega^c$ for exterior problems. By the vector $\mathbf{u}_{N_{\mathrm{dof}}}$ we denote the evaluation of the approximate potential in all these points $\boldsymbol{x}_i$. As we now consider an exterior problem, this potential evaluation afterwards is performed in several points $\boldsymbol{x}_i$ which are located on a sphere surrounding our geometry.

For the following discussion, we will use the pair $(l_1, l_2)$ to indicate the accuracy shift $l_1$ chosen in the growing routine and the shift $l_2$ chosen for solving the system. We fix the over-all shift by $l_1 + l_2 = 5$ and perform our computations for different combinations of $l_1$ and $l_2$, i.e. $(1,4)$, $(2,3)$, $(3,2)$ $(4,1)$ and $(5,0)$. Notice that we deliberately leave out the combination $(0,5)$. The bandwidth parameter $a$ is fix and set to $a = 2.5$. The coarsening constant has to chosen small enough in order to ensure optimal complexity, see [46]. Nonetheless, first experiments showed that the complexity seems to be basically the same if the computations are performed without coarsening. For this reason, we choose the rather large coarsening constant $\theta = 0.9$ for all the subsequent numerical experiments. These setting are used thoughout this chapter unless otherwise stated.

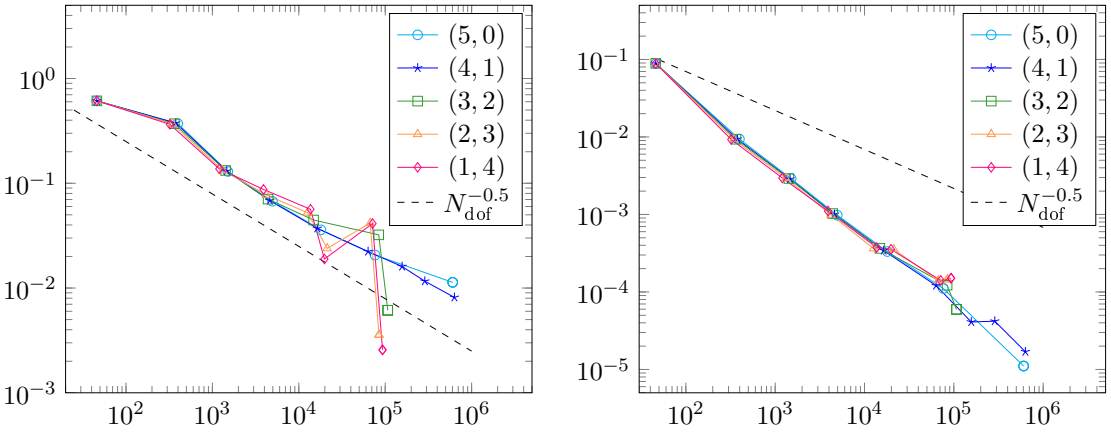



Figure 5.1: Norm of the residual (left) and potential error (right).

The right plot in Figure 5.1 displays the convergence of the maximal potential error with respect to the degrees of freedom together with the reference line $N_{\text{dof}}^{-0.5}$. We observe that all of the chosen accuracies $l_2 \in 0, \ldots, 4$ are high enough to guarantee a smooth convergence. Next, consider the left plot in Figure 5.1, which shows the estimated residual versus the degrees of freedom. Therein, we notice that the combinations $(1, 4)$, $(2, 3)$, and $(3, 2)$ cause some oscillations in the convergence. In contrast, the combinations $(4, 1)$ and $(5, 0)$ give a stable convergence of both, the residual and the potential.

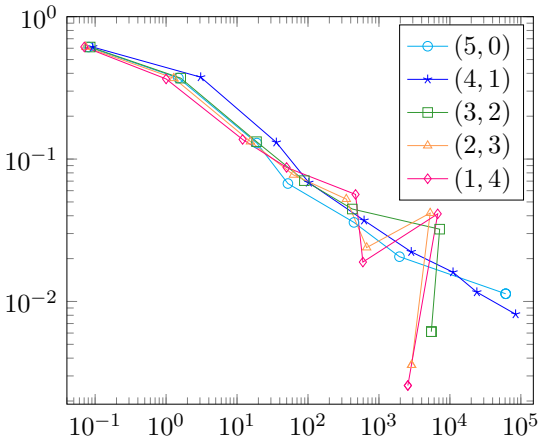



Figure 5.2: Norm of the residual versus computation time.

There remains the question on which of the two combinations $(4, 1)$ and $(5, 0)$ is better. To this end, let us consider the computation time as well. In Figure 5.2, one finds the computation time versus the the norm of the residual for all combinations under consideration. We observe that in the beginning for few degrees of freedom there is no big difference. Subsequently, the unstable combinations $(1, 4)$, $(2, 3)$ and $(3, 2)$ start to oscillate. This behaviour can be explained by not only looking at the degrees of freedom after coarsening, but also at the degrees of freedom used in the prediction step, which are not taken into

consideration in Figure 5.1. For the two combinations $(4,1)$ and $(5,0)$ we observe a similar over-all computation time, which is why we finally choose the combination $(5,0)$, as this combination is more accurate one.

## 5.2 Laplace Problems Solved by the Single Layer Operator

Let us now consider the Laplacian inside a given domain $\Omega$ with Dirichlet boundary conditions:

$$\begin{aligned} \Delta u &= 0 \qquad \text{in } \Omega, \\ u &= f \qquad \text{on } \Gamma. \end{aligned} \tag{5.4}$$

After this problem is converted into a boundary integral equation of the form (1.3), see Chapter 1, we use our adaptive wavelet method to compute the unknown density $\rho$ on the boundary $\Gamma$. With its help we evaluate the potential, given by equation (1.5), in several points inside of the domain.

For the following experiments, we choose in (5.4) different domains $\Omega$ as well as various right-hand sides $f$ on $\Gamma$. For the first and second example, we use $f = 1$ and we perform the computations on Fichera's vertex as well as a crankshaft geometry. In a third example, we choose the restriction $f = u|_\Gamma$ of the polynomial $u(\boldsymbol{x}) = 4x_1^2 - 3x_2^2 - x_3^2$ with $\boldsymbol{x} = (x_1, x_2, x_3)$ as Dirichlet data. This experiment is performed on a gearwheel geometry. All these surfaces are represented by patches as introduced in Section 2.5. The three geometries under consideration are illustrated in Figure 5.3.

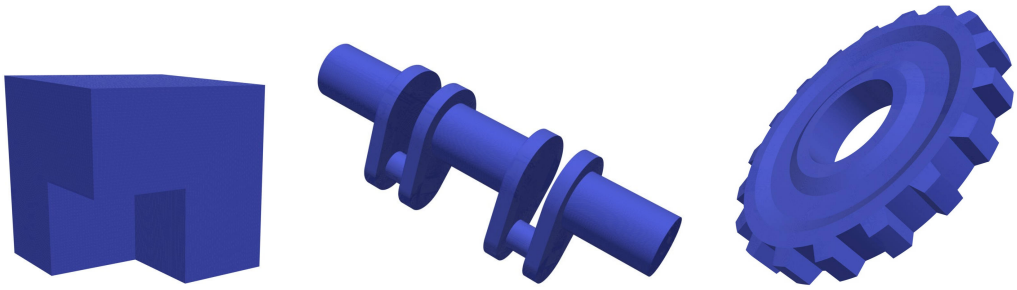


Figure 5.3: Fichera vertex (left), crankshaft (middle) and gearwheel (right).

In the rest of this thesis, whenever we present our numerical results, we use a table to display the numbers. Thereby, the outline of the tables is always the same. The numbers given in the first and second column (in this order) are the degrees of freedom $N_{\mathbf{res}}$ after prediction and the the degrees of freedom $N_{\mathrm{dof}}$ after coarsening. The third column contains the norm of the estimated residual $\|\mathbf{res}\|$ and the fourth column contains the error of the potential, which is again denoted by $\|\mathbf{u} - \mathbf{u}_{N_{\mathrm{dof}}}\|_\infty$. With $\mathbf{u} = [u(\boldsymbol{x}_i)]_i$ we denote again the exact evaluation of the function $u$ and with $\mathbf{u}_{N_{\mathrm{dof}}}$ the approximate evaluation of the potential. This time, we evaluate the approximate potential in multiple points $\boldsymbol{x}_i$ inside the domain $\Omega$. The fifth column (nnz (%)) shows us the percentage of non-zero entries in

the system matrix after compression. Finally, the last column contains the time in seconds used for one step of the code, that is the time for growing the tree and estimating the residual until the terminating condition is met, together with assembling and solving the system, but without evaluating the potential.

The first row of such a table, however, will contain only partial information and has to be interpreted as follows. We initialise our computation for the scaling functions on each patch. For these ansatz functions, we assemble the system matrix and the right-hand side and subsequently solve the system of linear equations in order to have a first approximation to the desired density. At this point, we cannot yet compute the residual, which can be done only when we have found the prediction set. Also, we do not list the time of this initialisation step, since it takes place before the growing loop. However, we can already use the first approximation to the density to evaluate the potential and to compute its maximal error. For all following rows, we have at hand the approximate density from the previous step. This density is then used for prediction, after which there are $N_{\mathbf{res}}$ degrees of freedom. On this enlarged set of basis functions, we estimate the residual. After coarsening the residual, there remain $N_{\mathrm{dof}}$ degrees of freedom for which the system of linear equations is solved, providing a new solution for the next step. For the same number of degrees of freedom, we finally evaluate the potential to compute the maximal potential error.

### 5.2.1 Fichera's Vertex

Let us present the results for the interior Dirichlet problem, solved on the Fichera vertex for the chosen constant right-hand side. As mentioned before, we initialise the computation by assembling and solving the system of linear equations for the 12 patches of the Fichera vertex and the subsequent potential evaluation.

| $N_{\mathbf{res}}$ | $N_{\mathrm{dof}}$ | $\|\mathbf{res}\|$ | $\|\mathbf{u}-\mathbf{u}_{N_{\mathrm{dof}}}\|_\infty$ | nnz (%) | time (s) |
|---|---|---|---|---|---|
| – | 12 | – | $6.09{\cdot}10^{-2}$ | 100 | – |
| 48 | 38 | $6.07{\cdot}10^{-2}$ | $3.58{\cdot}10^{-2}$ | 100 | 0.11 |
| 726 | 231 | $1.01{\cdot}10^{-1}$ | $9.60{\cdot}10^{-3}$ | 77.7 | 1.20 |
| 1788 | 825 | $4.64{\cdot}10^{-2}$ | $1.93{\cdot}10^{-3}$ | 30.5 | 6.16 |
| 5575 | 2005 | $2.64{\cdot}10^{-2}$ | $3.27{\cdot}10^{-4}$ | 10.4 | 12 |
| 14242 | 4708 | $1.60{\cdot}10^{-2}$ | $1.95{\cdot}10^{-4}$ | 4.40 | 48 |
| 39533 | 11311 | $1.06{\cdot}10^{-2}$ | $1.71{\cdot}10^{-4}$ | 1.63 | 158 |
| 101829 | 27497 | $6.38{\cdot}10^{-3}$ | $5.41{\cdot}10^{-5}$ | 0.62 | 572 |
| 244680 | 66815 | $3.78{\cdot}10^{-3}$ | $2.90{\cdot}10^{-5}$ | 0.26 | 1721 |
| 584032 | 150919 | $2.43{\cdot}10^{-3}$ | $7.32{\cdot}10^{-6}$ | 0.12 | 5034 |
| 1302958 | 353044 | $1.46{\cdot}10^{-3}$ | $5.03{\cdot}10^{-6}$ | 0.05 | 17364 |
| 2973431 | 779195 | $9.31{\cdot}10^{-4}$ | $2.04{\cdot}10^{-6}$ | 0.02 | 63883 |
| 6577954 | 1663259 | $5.94{\cdot}10^{-4}$ | $8.83{\cdot}10^{-7}$ | 0.01 | 234206 |

Table 5.1: Results for Fichera's vertex with the constant right-hand side.

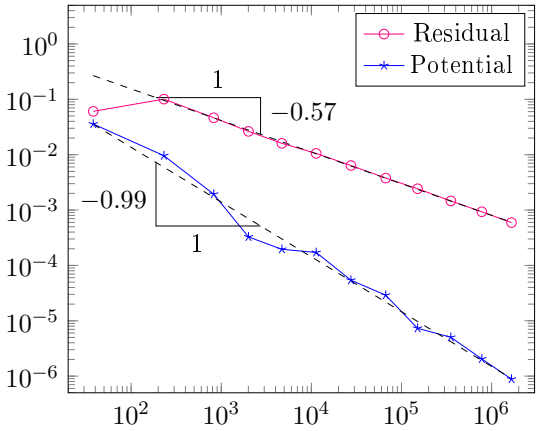



Figure 5.4: Residual and potential error for the single layer operator and Fichera's vertex.

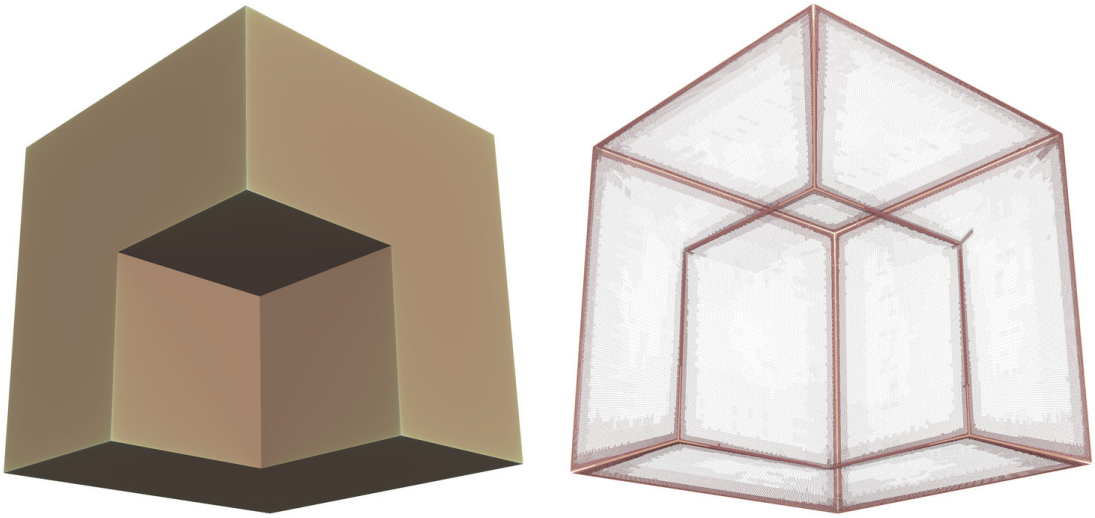


Figure 5.5: Solution and refinement on Fichera's vertex with constant right-hand side.

To ease the interpretation of the results, we shall represent our data by a graph. In Figure 5.4 we find, in a log-log plot, the number of degrees of freedom after coarsening $N_{\mathrm{dof}}$ versus the residual $\|\mathbf{res}\|$ and the potential error $\|\mathbf{u} - \mathbf{u}_{N_{\mathrm{dof}}}\|_\infty$, respectively. Notice that we omitted the data in the first line of the table. We observe that the rate of convergence for the residual is $N_{\mathrm{dof}}^{-0.5}$, even slightly better. The highest possible rate that can be achieved is $N_{\mathrm{dof}}^{-0.75}$. However, since we have a geometry with edges and corners, this rate is not met due to considering only an isotropic setting since the anisotropic singularities which appear at the edges cannot be resolved, see [74]. The convergence rate for the potential is considerably better, namely approximately $N_{\mathrm{dof}}^{-1}$, caused by super-convergence effects of the potential evaluation, see equation (5.3), as we are dealing with a linear output functional. From having a look back to the Table 5.1, we can clearly see that the percentage of entries in the system matrix decreases drastically and only very few entries remain.

To conclude the presentation of the results for Fichera's vertex, we draw the final density $\rho$

which has been computed in the left plot of Figure 5.5. Furthermore, we want to illustrate the behaviour of the adaptive code by drawing the final refinement. It is found in the right plot of Figure 5.5. Since we would not be able to see the refinement by drawing the grid which results from collating all elements with active wavelets, this picture was produced in the following way: After the code has terminated, we assign to each active wavelet the value $2^j$, with $j$ being the level of the wavelet. As a consequence, smaller wavelets get assigned a larger value. The picture above is thus to be interpreted as: First, each representative element of an active wavelet is visualised by plotting its four vertices. This means that more wavelets are located in areas with more points, which results in a darker colouring. Second, by looking at the geometry's edges more closely, one notices that at their center the colour gets lighter again. This change in colour is used to encode even finer elements in this area. As we consider the left plot of Figure 5.5, we observe that the most wavelets are added along the edges and at the vertices of the geometry. This behaviour is to be expected, as the density $\rho$ features, for a smooth right-hand side, the singularities of the interior and exterior problem for an indirect formulation. Another behaviour to be noticed at this point is that the edges towards the reentrant corner are not refined. This is unexpected in the first moment, but is in accordance with the left plot of Figure 5.5, where we notice that the solution does not feature singularities at any of these edges.

### 5.2.2 Crankshaft

As a second example, we consider the interior Dirichlet problem for the Laplace equation (5.4) on a crankshaft geometry. We choose the same constant right-hand side $f = 1$ as for Fichera's vertex. Table 5.2 contains the results obtained by the adaptive code, the outline of which is in accordance with the previous example. The computation starts again at the patch level by assembling and solving the system of linear equations, the solution to which is then used to evaluate the potential in several points which are located inside the crankshaft. The surface of this geometry is represented by 142 patches.

| $N_{\mathbf{res}}$ | $N_{\mathrm{dof}}$ | $\|\mathbf{res}\|$ | $\|\mathbf{u} - \mathbf{u}_{N_{\mathrm{dof}}}\|_\infty$ | nnz (%) | time (s) |
|---|---|---|---|---|---|
| – | 142 | – | $1.52{\cdot}10^{-1}$ | 100 | – |
| 568 | 460 | $2.40{\cdot}10^{-1}$ | $1.82{\cdot}10^{-2}$ | 92.3 | 2.64 |
| 8708 | 2787 | $9.62{\cdot}10^{-2}$ | $1.67{\cdot}10^{-2}$ | 20.0 | 43 |
| 18244 | 9615 | $5.51{\cdot}10^{-2}$ | $1.10{\cdot}10^{-2}$ | 7.72 | 92 |
| 45875 | 22158 | $4.14{\cdot}10^{-2}$ | $3.40{\cdot}10^{-3}$ | 4.14 | 412 |
| 116656 | 45898 | $1.63{\cdot}10^{-2}$ | $6.25{\cdot}10^{-4}$ | 1.44 | 1931 |
| 285240 | 101526 | $9.99{\cdot}10^{-3}$ | $3.44{\cdot}10^{-4}$ | 0.51 | 3822 |
| 648562 | 226703 | $5.92{\cdot}10^{-3}$ | $1.17{\cdot}10^{-4}$ | 0.23 | 11482 |
| 1390562 | 489995 | $3.60{\cdot}10^{-3}$ | $1.19{\cdot}10^{-4}$ | 0.10 | 34874 |
| 3166884 | 1084467 | $2.25{\cdot}10^{-3}$ | $1.20{\cdot}10^{-4}$ | 0.04 | 106941 |

Table 5.2: Results for the crankshaft with the constant right-hand side.

For visualisation, we plot again the convergence curves for both, the potential error and the norm of the residual, in logarithmic scale, see Figure 5.6. Observe that the rate of convergence for the norm of the residual is this time slightly better than for the previous example. Still, it does not reach the maximal order $N_{\mathrm{dof}}^{-0.75}$ either, as the crankshaft geometry also features edges and corners. The error of the potential is also slightly better than the rate of convergence for the norm of the residual, however, the difference between the two rates of convergence is smaller than in the previous example.

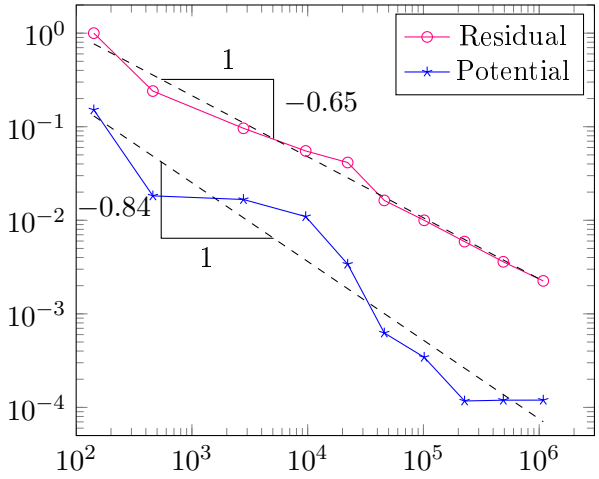



Figure 5.6: Residual and potential error for the single layer operator and the crankshaft.

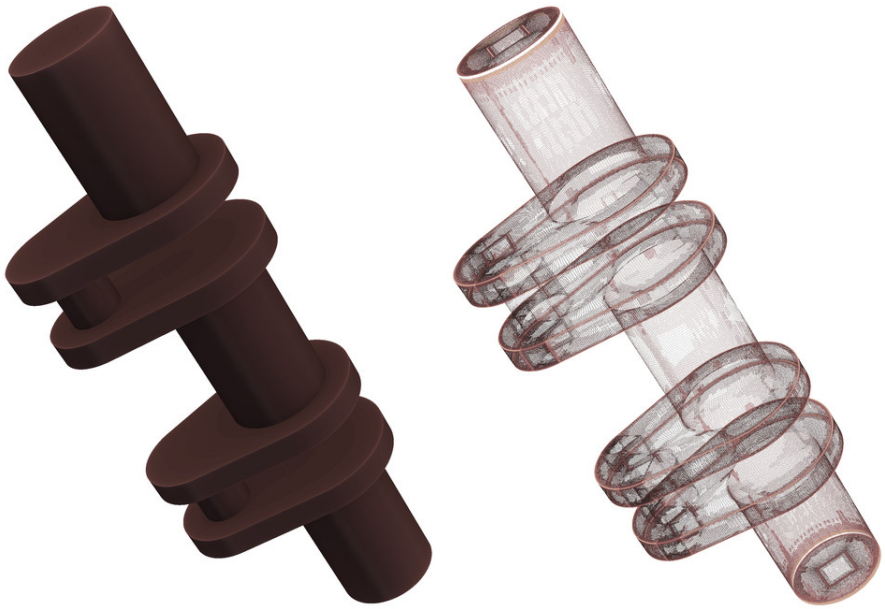


Figure 5.7: Solution (left) and refinement (right) on the crankshaft with constant right-hand side.

Finally, to conclude the results on the crankshaft, we visualise again the density $\rho$ and the refinement which is produced by the adaptive code by assigning the factor $2^j$ for each active wavelet. The left plot in Figure 5.7 shows the density and the right plot in Figure 5.7 the refinement. We observe that the adaptive code refines again towards the edges of the geometry.

### 5.2.3 Gearwheel

For the last experiment of the current section, we consider the geometry of a gearwheel with 18 teeth. This geometry is more demanding than the crankshaft and is represented by 502 patches. Recall that we choose a different right-hand side, i.e. the restriction $f = u|_\Gamma$ of the polynomial $u(\boldsymbol{x}) = 4x_1^2 - 3x_2^2 - x_3^2$. The analytical solution of the problem is then given by the polynomial $u(\boldsymbol{x})$. The gearwheel is much more complex than Fichera's vertex or the crankshaft, which is why it requires a different set of parameters than those suggested in Section 5.1. We choose here the compression parameter $a = 5$ and the layer shift $(l_1, l_2) = (5, 5)$.

| $N_{\mathbf{res}}$ | $N_{\mathrm{dof}}$ | $\|\mathbf{res}\|$ | $\|\mathbf{u} - \mathbf{u}_{N_{\mathrm{dof}}}\|_\infty$ | nnz (%) | time (s) |
|---|---|---|---|---|---|
| – | 504 | – | 2.54 | 100 | – |
| 2016 | 1716 | 6.35 | $8.98{\cdot}10^{-1}$ | 98.5 | 17 |
| 5084 | 4191 | 6.72 | $1.59{\cdot}10^{-1}$ | 55.2 | 38 |
| 6290 | 5605 | 3.04 | $7.82{\cdot}10^{-2}$ | 47.2 | 51 |
| 16384 | 10640 | 2.68 | $2.54{\cdot}10^{-2}$ | 28.4 | 151 |
| 29340 | 15999 | 1.92 | $1.75{\cdot}10^{-2}$ | 19.6 | 271 |
| 35415 | 20896 | 1.14 | $1.24{\cdot}10^{-2}$ | 17.0 | 385 |
| 41162 | 33010 | $3.72{\cdot}10^{-1}$ | $7.13{\cdot}10^{-3}$ | 13.2 | 721 |
| 136946 | 57152 | $8.54{\cdot}10^{-1}$ | $2.29{\cdot}10^{-3}$ | 10.1 | 3812 |
| 144262 | 101764 | $2.37{\cdot}10^{-1}$ | $3.02{\cdot}10^{-3}$ | 4.47 | 3554 |
| 319863 | 142179 | $4.31{\cdot}10^{-1}$ | $1.69{\cdot}10^{-3}$ | 3.78 | 10234 |
| 366229 | 254617 | $1.37{\cdot}10^{-1}$ | $1.32{\cdot}10^{-3}$ | 1.73 | 11844 |
| 929944 | 336611 | $2.86{\cdot}10^{-1}$ | $4.25{\cdot}10^{-4}$ | 1.83 | 45941 |
| 786306 | 597850 | $4.73{\cdot}10^{-2}$ | $3.85{\cdot}10^{-4}$ | 0.75 | 35142 |
| 2775537 | 811936 | $1.93{\cdot}10^{-1}$ | $6.85{\cdot}10^{-5}$ | 0.39 | 228111 |
| 1808460 | 1320772 | $2.97{\cdot}10^{-2}$ | $6.92{\cdot}10^{-5}$ | 0.16 | 104015 |

Table 5.3: Results for the gearwheel with the polynomial right-hand side.

In Table 5.3, we find the data produced by the adaptive code for the gearwheel geometry, where the first line has again to be interpreted specially. For this example, we observe a rate for the norm of the residual which seems to be approximately the optimal rate $N_{\mathrm{dof}}^{-0.75}$, although there are edges and vertices in the geometry. The error of the potential is even better and behaves again much better compared with the norm of the residual. We

visualise again the density $\rho$ on the gearwheel, which is found in the left plot of Figure 5.9 as well as the refinement behaviour of the adaptive code in the right plot of Figure 5.9. Notice that many wavelets are added on the the edges and the vertices again.

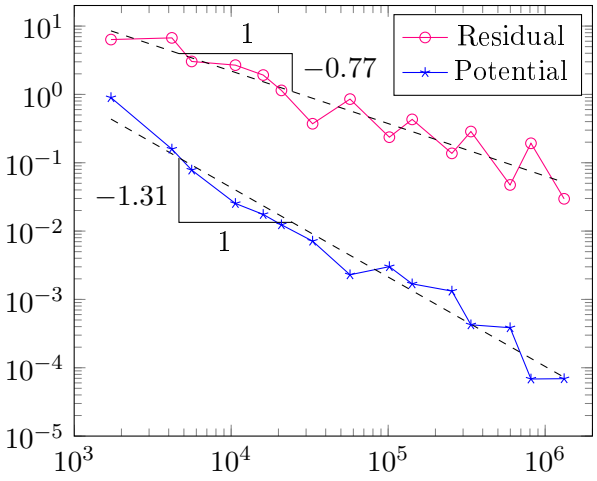



Figure 5.8: Residual and potential error for the single layer operator and the gearwheel.

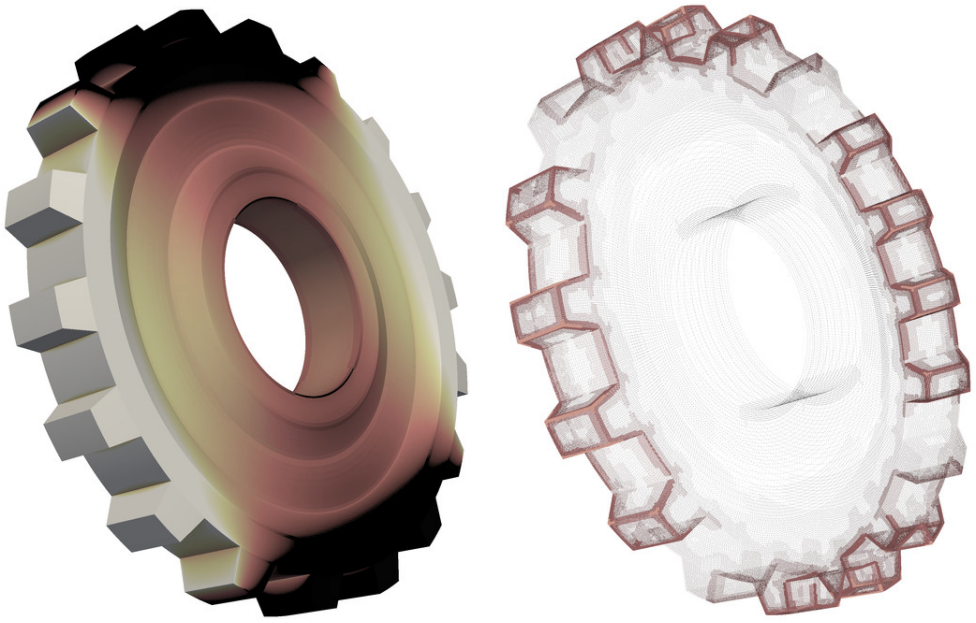


Figure 5.9: Solution and refinement on the gearwheel with polynomial right-hand side.

This example concludes the section for the solution of the Laplace equation by the means of the single layer potential. We see that the developed adaptive code works nicely for solving the Laplace equation, independently of the complexity of the domain. The number of non-zero entries in the matrix becomes indeed very small as the number of degrees of freedom increases, being one reason for being able to perform such computations in the first

place. Still, the computation time does not yet scale perfectly linearly in with respect to $N_{\mathrm{dof}}$. We observe that, for an increasing number of degrees of freedom, the administration the adaptive structures starts playing a role as well and there is still place for improvement. However, these structures are another reason that we are even able to perform adaptive computations on such a scale. Observe that the code is able to compute the adaptive solution up to more than one million wavelets still in reasonable time. There is no way to achieve a refinement of this degree in a uniform setting. Aside from the computability, there are situations for non-smooth right-hand sides, where uniform refinement would only give half the rate of adaptive refinement. We will encounter such examples in the following section.

## 5.3 Laplace Problems Solved by the Double Layer Operator

In this section, we will present our results for the interior Laplace equation solved by the double layer operator, defined by (1.7), for the unknown density $\rho$. For our first experiment, we consider again Fichera's vertex and the a smooth right-hand side $f = u|_\Gamma$ with $u(\boldsymbol{x}) = 4x_1^2 - 3x_2^2 - x_3^2$. We choose our parameters $a = 2.5$ and the layer shift $(l_1, l_2) = (5, 0)$ in accordance with our parameter study.

| $N_{\mathbf{res}}$ | $N_{\mathrm{dof}}$ | $\|\mathbf{res}\|$ | $\|\mathbf{u} - \mathbf{u}_{N_{\mathrm{dof}}}\|_\infty$ | nnz (%) | time (s) |
|---|---|---|---|---|---|
| – | 12 | – | | 100 | – |
| 48 | 37 | $5.93 \cdot 10^{-1}$ | $4.21 \cdot 10^{-1}$ | 100 | 0.14 |
| 681 | 298 | $3.73 \cdot 10^{-1}$ | $3.43 \cdot 10^{-2}$ | 74.0 | 2.87 |
| 2010 | 1234 | $1.33 \cdot 10^{-1}$ | $9.27 \cdot 10^{-3}$ | 32.1 | 28 |
| 18494 | 7821 | $6.69 \cdot 10^{-2}$ | $2.96 \cdot 10^{-3}$ | 4.68 | 484 |
| 101291 | 47477 | $2.84 \cdot 10^{-2}$ | $1.42 \cdot 10^{-3}$ | 0.85 | 3786 |
| 400552 | 215932 | $1.17 \cdot 10^{-2}$ | $1.80 \cdot 10^{-3}$ | 0.22 | 23635 |
| 1204802 | 717800 | $4.99 \cdot 10^{-3}$ | $1.79 \cdot 10^{-3}$ | 0.07 | 65891 |

Table 5.4: Results for Fichera's vertex and a polynomial right-hand side.

In Table 5.4, we find the output of the adaptive code, the outline of which is the same as for the previous tables. The convergence lines for the norm of the residual and the maximal error of the potential are drawn into the separate Figure 5.10. The highest possible rate of convergence for the norm of the residual attainable in the present setting is $N_{\mathrm{dof}}^{-0.5}$. Indeed this rate of convergence is observed, with the rate of convergence of the potential error being only marginally better.

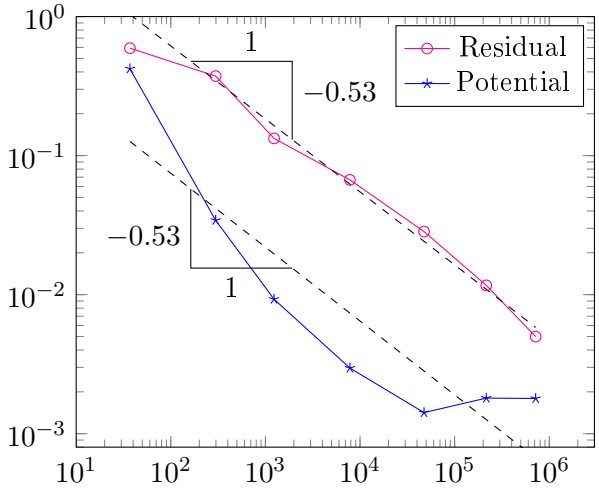



Figure 5.10: Residual and potential error for the double layer operator and Fichera's vertex.

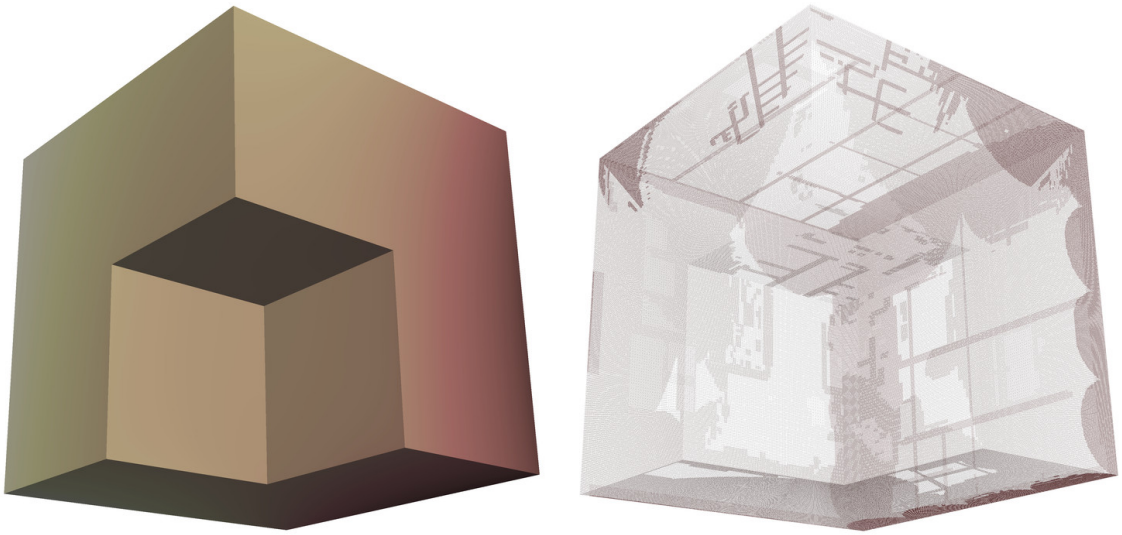


Figure 5.11: Solution (left) and refinement (right) on Fichera's vertex for the Laplace equation with a smooth right-hand side.

Though the adaptive code gives the desired rate of convergence of $N_{\mathrm{dof}}^{-0.5}$, we notice that the codes fares differently than for the examples involving the single layer operator given earlier. Let us consider the two plots in Figure 5.11. In the left plot, we display again the density and, in the right plot, the refinement behaviour of the algorithm. We notice that the code does not refine towards the edges and corners as it was the case for the single layer operator. Indeed, the refinement corresponds basically to a uniform refinement. Thus, it seems that solving the Laplace equation by means of the double layer operator does not pay off for a smooth right-hand side, even for a geometry featuring edges and vertices. Let us forestall that this behaviour can also be observed for the low-frequency Helmhotz

equation, which inspires the experiment in the following section on the exterior Helmholtz equation.

Before we turn towards the Helmholtz equation, let us first consider another experiment for solving the interior Laplace equation by means of the double layer operator on Fichera's vertex. We will consider Dirichlet data of the form

$$f(r) = \frac{1}{r^\alpha}$$

for $\alpha = 0.5$ and $\alpha = 0.75$ and with $r$ denoting the distance to a selected point $\boldsymbol{p}$. Notice, that this function is singular for $r = 0$. Thus, we choose the point $\boldsymbol{p}$, such that the singularity of the right-hand side lies on the surface of our domain. More precisely, for the Fichera vertex, this shall be the point where the reentrant corner is located, i.e. the point $\boldsymbol{p} = (0.5, 0.5, 0.5)$. Following the theory presented in [24], we can, for this right-hand side, expect a rate of convergence of least $N_{\text{dof}}^{-(1-\alpha)}$ when using an adaptive scheme. In contrast, we expect only half the rate $N_{\text{dof}}^{-(1-\alpha)/2}$ when using uniform refinement. To be more precise, in [24] it has been shown that the Besov regularity of the solution $\rho$ is twice as high as its Sobolev regularity for the integral operator under consideration.

Let us present the results for solving the interior Laplace equation using the double layer operator on Fichera's vertex for $\alpha = 0.5$. In Figure 5.12, we visualise these results. We plot the norm of the residual for both, uniform and adaptive refinement, into a log-log plot. The norm of the residual for uniform refinement was obtained as follows. For a fixed level of uniform refinement, we performed a prediction step analogously to adaptive refinement. On this enlarged set, we estimated the norm of the residual. Subsequently, we performed one standard step of uniform refinement.

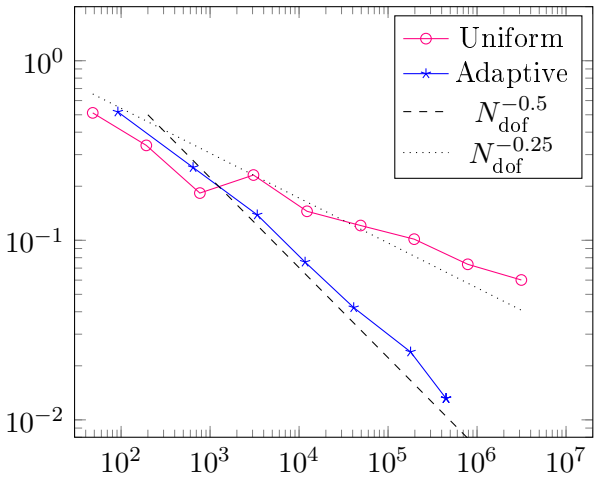



Figure 5.12: Norm of the residual for adaptive and uniform refinement for $\alpha = 0.5$.

We expect a diminished rate of convergence of $N_{\text{dof}}^{-(1-\alpha)/2} = N_{\text{dof}}^{-0.25}$ in the case of uniform refinement. In our results, we observe that the rate seems to be even slightly worse than

that. In case of adaptive refinement, we observe a rate $N_{\mathrm{dof}}^{-0.5}$, which is what we expect. Another observation can be made by comparing the number of degrees of freedom which are necessary in order to compute the approximate density for uniform refinement and for adaptive refinement. The norm of the residual is about $6 \cdot 10^{-2}$ for uniform refinement with more than 3 million degrees of freedom. For adaptive refinement, we obtain a norm of the residual of about $4.2 \cdot 10^{-2}$ already for approximately 40 000 degrees of freedom, which is quite impressive.

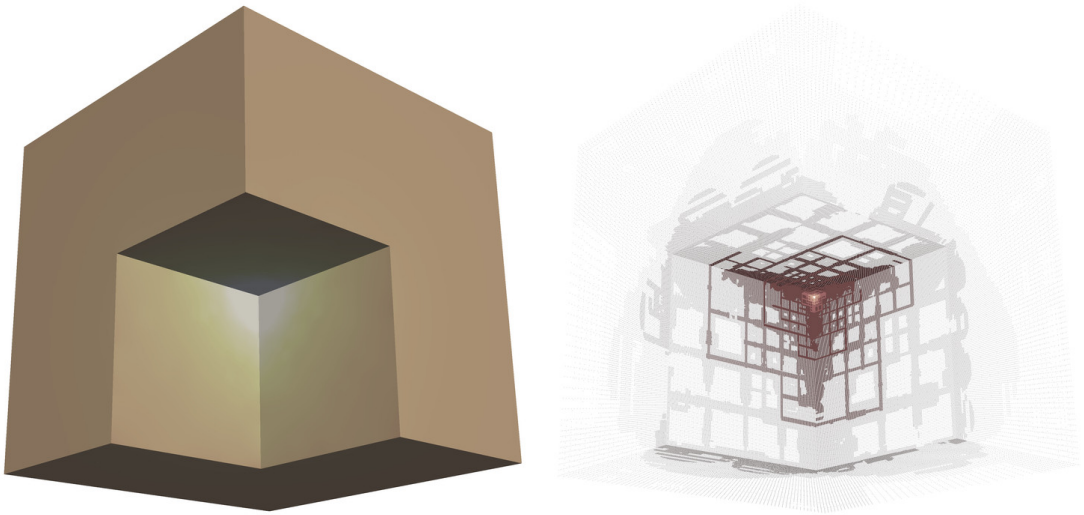


Figure 5.13: Density and refinement on Fichera's vertex for $\alpha = 0.5$.

In the left plot of Figure 5.13, we find the density and in the right plot of Figure 5.13 the refinement produced by the adaptive wavelet method. It is clearly visible that the adaptive method refines towards reentrant corner, where the right-hand side has its singularity. We also observe an interesting pattern in form of a grid in the refinement around the corner. This is an artefact and it can be reduced by choosing a smaller threshold parameter (meaning, that a larger portion of the residual is coarsened). Although the effect can be reduced, it never vanishes completely since it comes due to the large support of the wavelets.

Next, let us present the results for $\alpha = 0.75$. For this choice, we expect the adaptive wavelet method to converge at least at a rate $N_{\mathrm{dof}}^{-(1-\alpha)} = N_{\mathrm{dof}}^{-0.25}$. For uniform refinement, we expect again only half the rate $N_{\mathrm{dof}}^{-(1-\alpha)/2} = N_{\mathrm{dof}}^{-0.125}$.

Indeed, we can see that the adaptive wavelet method converges at the expected rate $N_{\mathrm{dof}}^{-0.25}$ or even slightly better. In comparison, for uniform refinement we observe the reduced rate $N_{\mathrm{dof}}^{-0.125}$. If we compare the norm of the residuals for both strategies, we observe again the superiority of the adaptive code. In order for uniform refinement to produce a value of $2.86 \cdot 10^{-1}$, it needs more that 3 million degrees of freedom. The adaptive code produces an error of $2.72 \cdot 10^{-1}$ with less than 15 000 degrees of freedom.

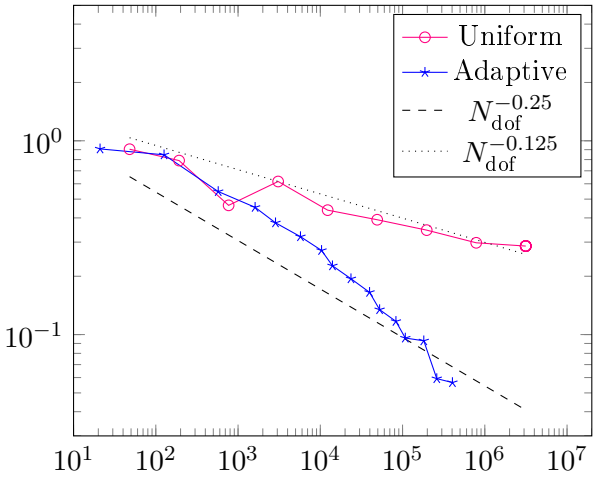



Figure 5.14: Norm of the residual for adaptive and uniform refinement for $\alpha = 0.75$.

Next, let us consider Figure 5.15 which displays the density in the left plot as well as the refinement in the right plot for the adaptive wavelet method for $\alpha = 0.75$. We observe again a strong refinement towards the reentrant corner, whereas most of the remaining surface stays coarse. In accordance to the previous results for $\alpha = 0.5$, we notice again the grid-like artefacts around the corner. They would be diminished when choosing a smaller threshold constant which forces the adaptive algorithm to discard more non relevant entries in the residual.

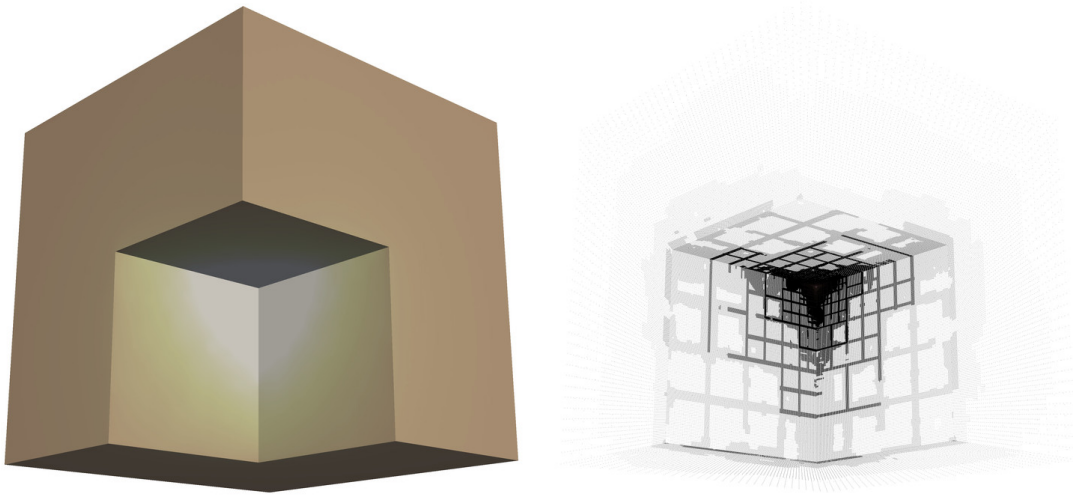


Figure 5.15: Solution and refinement on the Fichera vertex for $\alpha = 0.75$.

In conclusion, we see that the adaptive algorithm does not really pay off for integral equations solved by the double layer operator, if they feature a smooth right-hand side. In these situations, the adaptive algorithm does not seem to notice the edges and corners of the surface of the domain. However, for non-smooth right-hand sides, we observe that adaptiveness really does pay off. This observation goes in accordance with the theory developed in [24].

## 5.4 Exterior Helmholtz Problems

In this section, we will present numerical results in case of the exterior Helmholtz problem (5.1). According to equation (1.21), the combination $u = (\mathcal{D} - i\eta\mathcal{S})\rho$ of the acoustic single layer potential $\mathcal{S}$ and the acoustic double layer potential $\mathcal{D}$ can be used to avoid spurious modes, and leads to the second kind Fredholm boundary integral equation (1.21).

### 5.4.1 Cartoon Right-Hand Side

We will present an example for solving the exterior Helmholtz equation with Dirichlet boundary data for $\kappa = 1$. Recall the experiment in Section 5.3, where we observed that adaptivity does not pay off when the right-hand side is smooth, although the geometry under consideration has edges and vertices. The same can be observed for the Helmholtz equation. This motivates the following example.

Let us consider the sphere as geometry, which has a smooth surface such that the boundary integral operator offers the full regularity. As right-hand side, we choose a so-called cartoon function:

$$f(\boldsymbol{x}) = \begin{cases} 1, & \text{if} \quad \|\boldsymbol{x} - (0,0,1)\|^2 \leq 0.125, \\ 0, & \text{else.} \end{cases} \tag{5.5}$$

As such a functions can be approximated at a rate $N_{\text{dof}}^{-0.5}$, see [34], we expect that the density $\rho$ converges at the same rate. In contrast, we would expect only half the rate for uniform refinement, as this right-hand side is only in the Sobolev space $H^{1/2-\delta}$ with $\delta > 0$ arbitrarily small, see [24].

| $N_{\mathbf{res}}$ | $N_{\text{dof}}$ | $\|\mathbf{res}\|$ | nnz (%) | time (s) |
|---|---|---|---|---|
| – | 6 | – | 100 | – |
| 76 | 16 | $4.04{\cdot}10^{-1}$ | 100 | 0.14 |
| 1496 | 110 | $1.64{\cdot}10^{-1}$ | 30.77 | 7.83 |
| 3836 | 843 | $4.21{\cdot}10^{-2}$ | 12.74 | 27 |
| 25590 | 4813 | $1.99{\cdot}10^{-2}$ | 2.165 | 371 |
| 53172 | 13807 | $1.02{\cdot}10^{-2}$ | 1.022 | 658 |
| 223670 | 46892 | $7.56{\cdot}10^{-3}$ | 0.245 | 4344 |
| 637357 | 122770 | $4.57{\cdot}10^{-3}$ | 0.085 | 14512 |
| 1625857 | 331370 | $2.81{\cdot}10^{-3}$ | 0.033 | 66967 |
| 6211463 | 968764 | $2.08{\cdot}10^{-3}$ | 0.003 | 375731 |

Table 5.5: Results for the sphere with a discontinuous right-hand side.

In Table 5.5, we find the output of the adaptive wavelet method. As the analytical solution of the Helmholtz equation for the Dirichlet data $f$ is unknown, we cannot compute the

potential error. Figure 5.16 shows the convergence for the norm of the residual in comparison with the number of the degrees of freedom. Therein, we observe again a rate of convergence of $N_{\mathrm{dof}}^{-0.5}$. Also, we see a rate of convergence of $N_{\mathrm{dof}}^{-0.25}$ for uniform refinement.

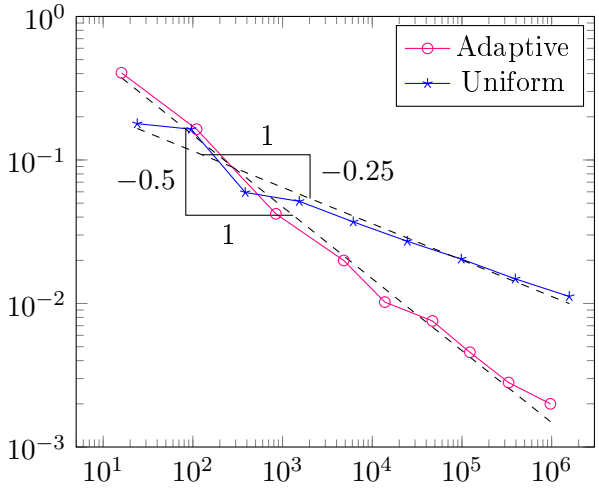



Figure 5.16: Norm of the residual in case of a discontinuous right-hand side for uniform and adaptive refinement.

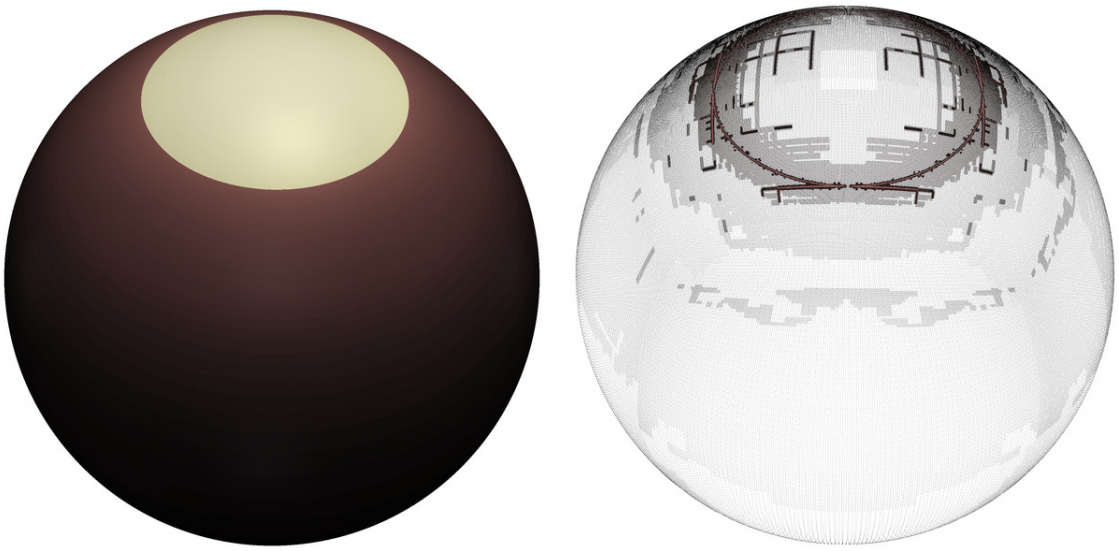


Figure 5.17: Solution (left) and refinement (right) on the sphere for the exterior Helmholtz equation with discontinuous right-hand side for the wavelets with three vanishing moments.

In the left plot of Figure 5.17, we find again the density on the sphere, where we can clearly see the jump in the density coming from the discontinuity of the right-hand side. In the right plot of Figure 5.17, we have visualised the refinement produced by the adaptive wavelet method. We observe that the refinement takes place exactly around the jump, as we expect, while the other parts stay very coarse.

### 5.4.2 Scattering Problems

Next, we will present numerical results for the scattering problem, with different wavenumbers $\kappa \geq 1$. The choice of larger wavenumbers has a direct effect on the sparsity of the system matrix, as the wavenumber has an influence on the compression of the matrix see, [65].

Let us recall the layout for a scattering problem. Let $u$ be the solution to the Helmholtz equation, where $u$ consists of an incident and a scattered wave, i.e. $u = u_s + u_i$. The incident wave $u_i$ is known and is of the form $\exp(i\kappa \boldsymbol{d}\boldsymbol{x})$, where $\boldsymbol{d}$ denotes the direction (it holds $\|\boldsymbol{d}\| = 1$), and the goal is to compute $u^s$. With $u^s$ given, the solution $u$ to the Helmholtz equation can be computed as well. In order to find $u$, we can either use a combination of the acoustic single layer operator and the acoustic double layer operator as described in equation (1.23), or we use the direct ansatz (1.26), which we repeat for convenience here

$$u(\boldsymbol{x}) = u^i(\boldsymbol{x}) - \int_\Gamma k(\boldsymbol{x}, \boldsymbol{y}) \frac{\partial u}{\partial \boldsymbol{n}}(\boldsymbol{y}) \mathrm{d}\sigma_y, \quad \boldsymbol{x} \in \Omega^c. \tag{5.6}$$

The unknown Neumann data $\frac{\partial u}{\partial \boldsymbol{n}}$ is obtained by solving the boundary integral equation

$$\left(\frac{1}{2}\mathbf{I} + \mathcal{D}^t - i\eta\mathcal{S}\right) \frac{\partial u}{\partial \boldsymbol{n}} = \frac{\partial u^i}{\partial \boldsymbol{n}} - i\eta u^i \quad \text{on } \Gamma. \tag{5.7}$$

For the following computations, we will solve the scattering problem by using the direct formulation (5.6), (5.7). As geometry, we consider a drilled cube. After solving the boundary integral equation (5.7), we have given the Neumann data which can be used to evaluate $u(\boldsymbol{x})$ according to equation (5.6).

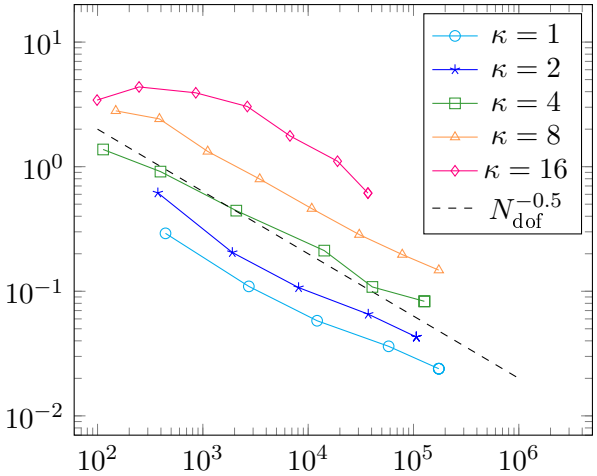



Figure 5.18: Norm of the residual for the adaptive wavelet method.

For the visualisation of the solution, we will compute the total field $u(\boldsymbol{x})$ and the scattered field $u^s(\boldsymbol{x})$ in the area $E = \{(x_1, x_2, x_3) \colon x_3 = 0 \text{ and } x_1, x_2 \in [-2.5, 2.5]\}$. We perform the

computations for $\kappa = 1$, 2, 4, 8 and 16.

Figure 5.18 shows the convergence history of the norm of the residual for each different value of $\kappa$. We observe a convergence of approximately $N_{\text{dof}}^{-0.5}$, independently of the chosen $\kappa$.

Similar to the observations in Section 5.3, we noticed again a diminished rate for uniform refinement for all chosen $\kappa$. We did not draw this into the Figure 5.18, in order not to overload it. Nevertheless, for $\kappa = 8$, we illustrate this behaviour in Figure 5.19. For uniform refinement we observe a rate of convergence of approximately $N_{\text{dof}}^{-0.25}$, which is only half the rate as for adaptive refinement.

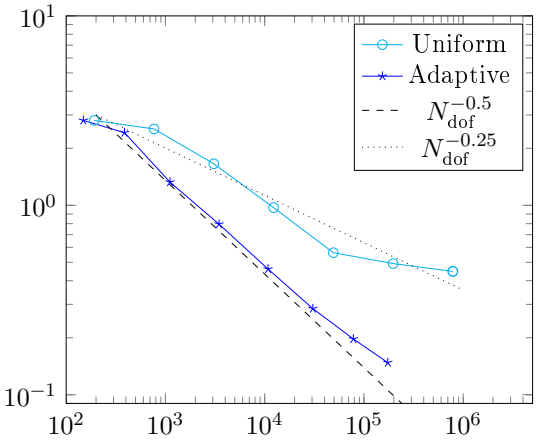



Figure 5.19: Norm of the residual for adaptive refinement and for uniform refinement.

As mentioned earlier, we draw the total field $u(\boldsymbol{x})$ and the scattered field $u^s(\boldsymbol{x})$ in the $x_1$-$x_2$-plane for $x_1, x_2 \in [-2.5, 2.5]$ for each chosen $\kappa = 1$, 2, 4, 8, and 16. This plane intersects the drilled cube, such that we can illustrate the pattern which is produced by the scattered wave. Thereby we chose the incident wave to travel into the direction of $(1, 1, 0)$. In addition to the plane, where $u^s(\boldsymbol{x})$ and $u(\boldsymbol{x})$ are evaluated, we also draw the scatterer in the pictures. In particular, we draw the refinement of the scatterer's surface, where a cluster of points appears in a darker colour first. By looking more closely at the corners and edges of the geometry, we see a lighter colouring, which again indicates even a stronger refinement.

In Figure 5.20, we see the scene for $\kappa = 1$. The top corner of the square is the point $(-2.5, -2.5)$, which means that the harmonic wave is travelling upwards. In the left plot of Figure 5.20, the scattered field $u^s(\boldsymbol{x})$ is seen and, in the right plot of Figure 5.20, the total field $u(\boldsymbol{x})$ is seen. For $\kappa = 1$, we do not observe yet an interesting scattering pattern, as the wavenumber is too small. On the other hand, we already observe that the adaptive wavelet method refines towards the edges and the vertices of the geometry again. This behaviour came unexpected at first, as we did not have any such refinements for the scattering problem solved by the indirect formulation. However, it makes sense, as we solve

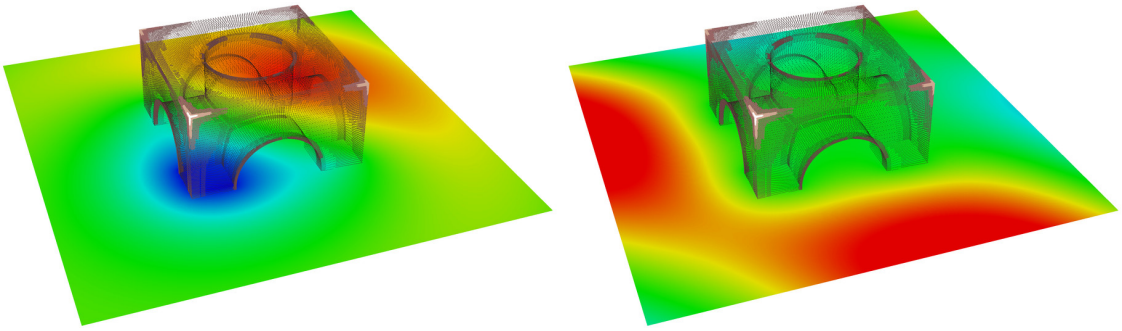


Figure 5.20: Scattered field (left) and total field (right) for $\kappa = 1$.

the scattering problem for the direct formulation featuring the Neumann data, which have a jump at the non-smooth parts of the geometry. In particular, we observe a refinement on the edges which are illuminated, i.e. the edges which face the incoming wave. On the edges which are at the back of the geometry refinement, the refinement is not as strong.

In Figure 5.21, we visualise the scattered field and the total field for $\kappa = 2$. The angles of the pictures are chosen equal to those of Figure 5.20 such that we can easily compare the situations. We observe that the wavenumber $\kappa = 2$ is still too small to have a noticeable scattering pattern. Also, we observe again the refinement along the illuminated edges in reference to the incoming wave.

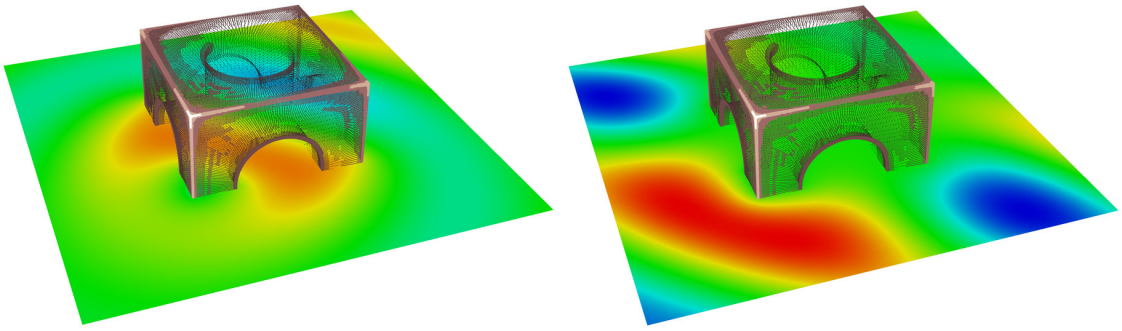


Figure 5.21: Scattered field (left) and total field (right) for $\kappa = 2$.

Figure 5.22 contains the scattered field (left) and total field (right) for the wavenumber $\kappa = 4$. Here, we observe that the wavenumber is chosen just large enough, such that for the first time the wave can enter the inner part of the drilled cube. We observe again the refinement towards the edges and vertices facing the incoming wave, with less refinement in these parts of the geometry which lies on the back of the cube.

In Figures 5.23 and 5.24, we draw the scattered field (left) and the total field (right) for the wavenumbers $\kappa = 8$ and 16, respectively. These wavenumbers are large enough in order to produce a beautiful scattering pattern. Also, we see that both waves can travel

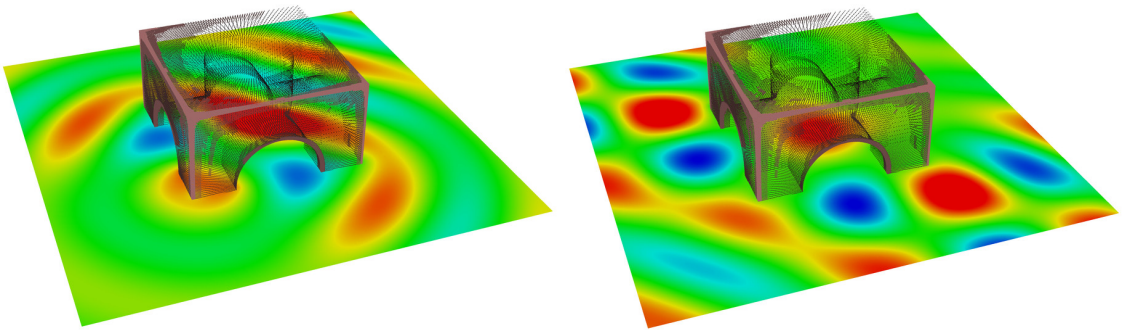


Figure 5.22: Scattered field (left) and total field (right) for $\kappa = 4$.

through the inner part of the drilled cube. We observe again the refinement towards the edges and vertices with more refinement of those parts of the drilled cube which are hit by the incoming wave. This can be seen very beautifully in Figure 5.25, showing the total field and the scatterer in top view. Notice the colour gradient in the form of a drop which forms around the hole of the cube.

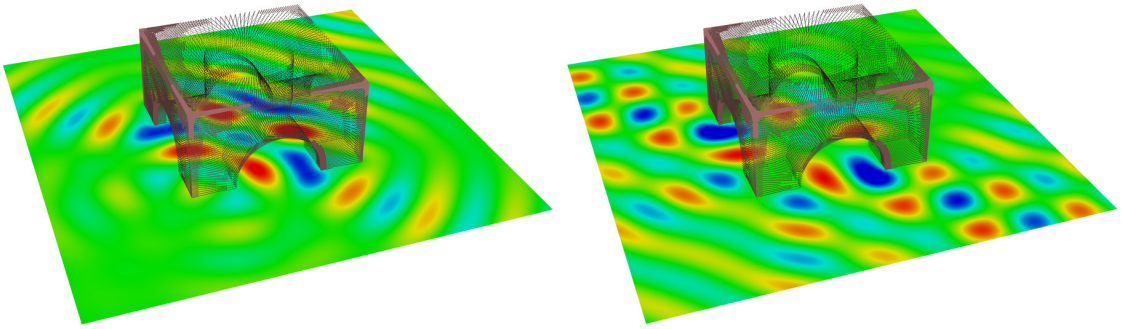


Figure 5.23: Scattered field (left) and total field (right) for $\kappa = 8$.

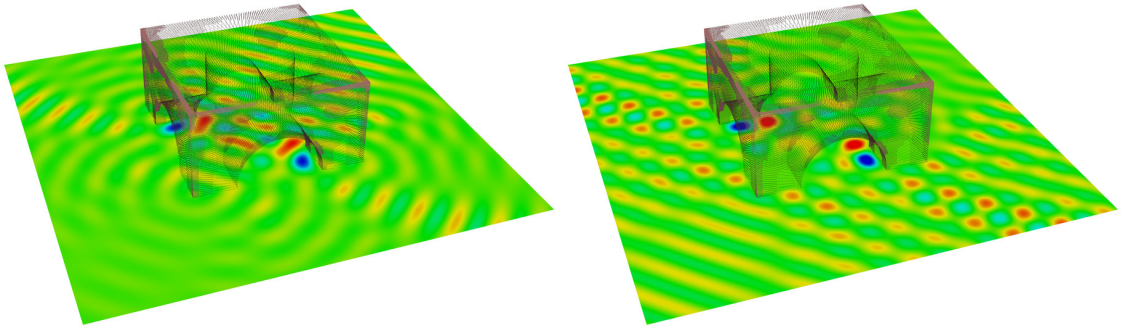


Figure 5.24: Scattered field (left) and total field (right) for $\kappa = 16$.

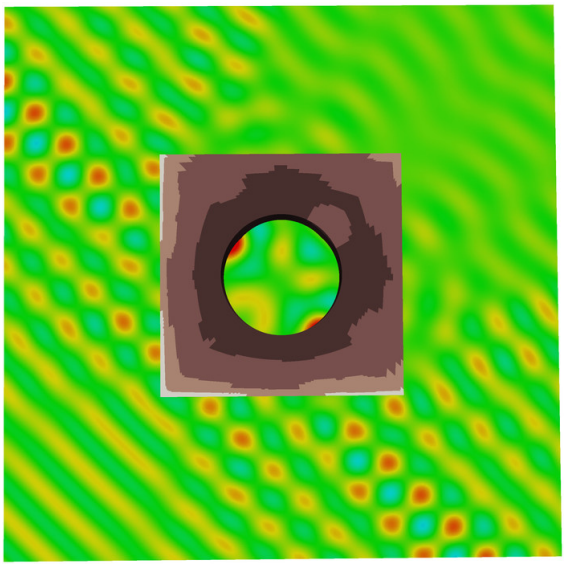


Figure 5.25: Top view of the scattered field for $\kappa = 16$.

In conclusion, we can say that the adaptive wavelet algorithm produces excellent results for all chosen wavenumbers $\kappa$. Adaptivity pays off especially for the direct ansatz, where the refinement towards the edges and vertices can be clearly observed. To achieve a similar accuracy with a uniform code, one would need much more degrees of freedom, which would not only take more time to compute, but may not be feasible any more as far as memory consumption is concerned.

# 6

# Goal-Oriented Error Estimation

In this chapter, we will be concerned with the solution of boundary integral equations by means of a goal-oriented adaptive wavelet method. In some previous examples, especially for the double layer operator for the Laplace equation and the Brakhage-Werner formulation for the Helmholtz equation, we have observed that an adaptive approach does have no advantage in comparison with a non-adaptive method. This is the case even if we consider a domain featuring edges and vertices.

However, for many applications one is not interested in the unknown density $\boldsymbol{\rho}$, but only in a continuous, linear output functional $g(\boldsymbol{\rho})$ of it. Approximating this new quantity of interest instead is referred to as goal-oriented method. By considering only an output functional of the density, one may be able to perform the computation with much less degrees of freedom.

## 6.1 Motivation and Background

There has been a large amount of research in the field of goal-oriented adaptive finite element methods, see e.g. [1, 5, 6, 31, 38, 76] and the references therein. Related results in the context of adaptive boundary element methods can be found in e.g. [40–42]. Our results will be mainly based on [40, 76]. Notice that an improved algorithm can be found in [31]. Nevertheless, we do not follow this approach since the proposed strategy is much more sophisticated and too intrusive.

The first observation, which can be made concerning the error $|g(\boldsymbol{\rho}) - g(\boldsymbol{\rho}_{\mathcal{J}})|$, is that it is bounded through

$$|g(\boldsymbol{\rho}) - g(\boldsymbol{\rho}_{\mathcal{J}})| red \leq \|g\| \, \|\boldsymbol{\rho} - \boldsymbol{\rho}_{\mathcal{J}}\|_{\ell^2(\mathcal{J}_\star)}$$

Thus, the error of the functional converges at least at the same rate as the error $\|\boldsymbol{\rho} - \boldsymbol{\rho}_{\mathcal{J}}\|_{\ell^2(\mathcal{J}_\star)}$ of the density. The next step is to introduce the dual (or adjoint) problem

$$\boldsymbol{A}'\boldsymbol{z} = \boldsymbol{g},$$

with $\boldsymbol{z}$ denoting the dual solution. After restricting the infinite index set $\mathcal{J}_\star$ to a finite subset $\mathcal{J} \subset \mathcal{J}_\star$ of cardinality $N$, we end up with the following system of linear equations on the index set $\mathcal{J}$ for the dual problem:

$$\boldsymbol{A}'_{\mathcal{J}}\boldsymbol{z}_{\mathcal{J}} = g(\boldsymbol{\rho}_{\mathcal{J}}).$$

With the dual problem taken into account, one arrives in view of Galerkin orthogonality at the estimate

$$|g(\boldsymbol{\rho}) - g(\boldsymbol{\rho}_{\mathcal{J}})| \lesssim \|\boldsymbol{\rho} - \boldsymbol{\rho}_{\mathcal{J}}\|_{\ell^2(\mathcal{J}_\star)} \, \|\boldsymbol{z} - \boldsymbol{z}_{\mathcal{J}}\|_{\ell^2(\mathcal{J}_\star)}.$$

If approximations to both, the primal and dual solutions, can be found, such that the error of the primal solution convergences at least at a rate of $N^{-s}$ and the dual solution converges at a rate of at least $N^{-t}$, then, according to [76], a rate of convergence of order $N^{-(s+t)}$ can be expected for the error of the output functional.

We consider here two different strategies of a goal-oriented adaptive wavelet method. One strategy is according to Mommer and Stevenson [76] and separately minimises the error of the primal problem and the error of the dual problem, respectively. Subsequently, the smaller of the underlying index sets is chosen for refinement. Another strategy is to use the union of the underlying index sets, which greatly simplifies the algorithm. In the following examples we will specify these two strategies and discuss their advantages and drawbacks.

We would like to mention at this point that the goal-oriented approach is not restricted to linear output functionals, but can also be extended to non-linear output functionals, see e.g. [1] and the references therein. However, for the sake of simplicity we are considering only a linear output functional here.

## 6.2 Refinement Strategies

As mentioned, we pursue two different strategies for which we start with the following setting. For an initial finite index set $\mathcal{J} \subset \mathcal{J}_\star$, the system of linear equations (3.3) is assembled and solved (for the primal problem and for the dual problem) with accuracy $\epsilon$. Subsequently, we call the routine GROW (outlined in Algorithm 1) for the primal problem, with the input parameters $\mathcal{J}$ and $\eta_{\text{primal}}$. The input parameter $\eta_{\text{primal}}$ is initialised at the beginning by an $\eta_{\text{init}}$ of our choice, and it is modified during the growing routine as outlined in Algorithm 1. After this routine terminates, we call GROW again for the dual problem. Similarly to the primal problem, we have a second parameter $\eta_{\text{dual}}$, which is also initialised at the beginning and modified during the growing routine. Inside GROW, the index set $\mathcal{J}$ is extended to $\mathcal{J}'_1$ for the primal problem. Likewise, the index set $\mathcal{J}$ is extended to $\mathcal{J}'_2$ for

the dual problem. This strategy is outlined in Algorithm 16.

**Algorithm 16:** Refinement based on the minimal index set.

Given: Initial index set $\mathcal{J}$, $\eta_{\text{init}}$;
Set: $\eta_{\text{primal}} = \eta_{\text{dual}} = \eta_{\text{init}}$;
Calculate: $\boldsymbol{\rho}_{\mathcal{J}} = \text{SOLVE}[\mathcal{J}, \epsilon]$;
Calculate: $\boldsymbol{z}_{\mathcal{J}} = \text{SOLVE}[\mathcal{J}, \epsilon]$;

**do**
  $\mathcal{J}_1^{'} = \text{GROW}\ [\mathcal{J}, \eta_{\text{primal}}]$;
  $\mathcal{J}_2^{'} = \text{GROW}\ [\mathcal{J}, \eta_{\text{dual}}]$;
  **if** $|\mathcal{J}_1^{'}| \leq |\mathcal{J}_2^{'}|$ **then**
    $\mathcal{J} = \mathcal{J}_1^{'}$;
  **else**
    $\mathcal{J} = \mathcal{J}_2^{'}$;
  **end**
  **for** $i = 0$ **to** $\#\mathcal{J}$ **do**
    Set: `wavelet[`$i$`]` $\rightarrow$ `layer =`
      max{ `wavelet[`$i$`]` $\rightarrow$ `layerPrimal, wavelet[`$i$`]` $\rightarrow$ `layerDual` };
  **end**
  Calculate: $\boldsymbol{\rho}_{\mathcal{J}} = \text{SOLVE}[\mathcal{J}, \epsilon \cdot \eta_{\text{primal}}]$;
  Calculate: $\boldsymbol{z}_{\mathcal{J}} = \text{SOLVE}[\mathcal{J}, \epsilon \cdot \eta_{\text{dual}}]$;
**end**

Inside each of the routines $\text{GROW}[\mathcal{J}, \eta_{\text{primal}}]$ and $\text{GROW}[\mathcal{J}, \eta_{\text{dual}}]$, the tree structured index set $\mathcal{J}$ is first sorted according to equation (4.3) and subsequently decomposed into layers, such that estimate (3.21) holds. As the routine GROW is performed separately for the primal problem and for the dual problem, a wavelet may receive a different layer associated with the primal problem or the dual problem, respectively. This is why we introduce two additional variables inside the `wavelet` structure, namely `int layerPrimal` and `int layerDual`. The variable `layerPrimal` is associated with the layer classification for the primal problem and the variable `layerDual` is associated with the layer classification for the dual problem. After the growing is complete for both, the primal problem and the dual problem, the smaller of both index sets $\mathcal{J}_1^{'}$ and $\mathcal{J}_2^{'}$ is chosen to be redefined as the new initial index set $\mathcal{J}$. We then set the layer of each wavelet in the final index set $\mathcal{J}$ to the maximum of both layers. After each wavelet has received the maximal layer, we assemble and solve the system of linear equations with respect to the index set $\mathcal{J}$.

The second strategy works, in principle, similar to the first strategy. The difference is that, after the primal and dual growing, the union of both sets $\mathcal{J} = \mathcal{J}_1^{'} \cup \mathcal{J}_2^{'}$ is taken as the new initial set $\mathcal{J}$. Also, we subsequently set the layer of each wavelet to the maximum of either `layerPrimal` or `layerDual`. As a consequence, we have to deal with more degrees of freedom by using this strategy. However, there is the possibility for a much more efficient realisation, which manages to assemble the system of linear equations only once in each

loop. This requires the modified routine GrowMod, which is outlined in Algorithm 17.

**Algorithm 17:** Modified growing routine GrowMod

Given: $\mathcal{J}$, $\eta_{\text{primal}}$, $\eta_{\text{dual}}$, $\boldsymbol{\rho}_{\mathcal{J}}$ and $\boldsymbol{z}_{\mathcal{J}}$;
**do**
Set: $\eta_{\text{primal}} = \eta_{\text{primal}}/2$, $\eta_{\text{dual}} = \eta_{\text{dual}}/2$ ;
sort$(\boldsymbol{\rho}_{\mathcal{J}}, \|\cdot\|_s)$;
sort$(\boldsymbol{z}_{\mathcal{J}}, \|\cdot\|_s)$;
**for** $i = 0$ **to** $\#\mathcal{J}$ **do**
Set: `wavelet[`$i$`]` $\rightarrow$ `layer =`
max{ `wavelet[`$i$`]` $\rightarrow$ `layerPrimal, wavelet[`$i$`]` $\rightarrow$ `layerDual` };
**end**
$\mathcal{J}' =$ Predict$[\mathcal{J}]$;
Calculate $\boldsymbol{r}_{(\mathcal{J}',\text{primal})} =$ Rhs$[\eta_{\text{primal}}] -$ Apply$[\eta_{\text{primal}}, \boldsymbol{A}, \boldsymbol{\rho}_{\mathcal{J}}]$;
Calculate $\boldsymbol{r}_{(\mathcal{J}',\text{dual})} =$ Rhs$[\eta_{\text{dual}}] -$ Apply$[\eta_{\text{dual}}, \boldsymbol{A}^t, \boldsymbol{z}_{\mathcal{J}}]$;
Set: $\mathfrak{r}_{\text{primal}} = \|\boldsymbol{r}_{(\mathcal{J}',\text{primal})}\|$, $\mathfrak{r}_{\text{dual}} = \|\boldsymbol{r}_{(\mathcal{J}',\text{dual})}\|$;
Calculate $\boldsymbol{r}_{(\mathcal{J}_1'',\text{primal})} =$ Coarse$[\boldsymbol{r}_{(\mathcal{J}',\text{primal})}, \theta]$ ($0 < \theta < 1$ threshold constant);
Calculate $\boldsymbol{r}_{(\mathcal{J}_2'',\text{dual})} =$ Coarse$[\boldsymbol{r}_{(\mathcal{J}',\text{dual})}, \theta]$;
Set: $\mathcal{J} = \mathcal{J}_1'' \cup \mathcal{J}_2''$;
**while** $\eta_{\text{primal}} \geq \mathfrak{r}_{\text{primal}}$ *AND* $\eta_{\text{dual}} \geq \mathfrak{r}_{\text{dual}}$;

Inside the routine GrowMod, we perform the following steps. Let $\boldsymbol{\rho}_{\mathcal{J}}$ and $\boldsymbol{z}_{\mathcal{J}}$ be the primal solution and the dual solution from a previous step. First, we sort the tree according to equation (4.3) for the primal solution $\boldsymbol{\rho}_{\mathcal{J}}$ and also for the dual solution $\boldsymbol{z}_{\mathcal{J}}$. Here, it can happen that a wavelet receives two different layers `layerPrimal` and `layerDual`. After the layer of each wavelet is again set to the maximum of the primal layer and the dual layer, we call the routine Predict. Since the layer is set to the maximum, we make sure that the routine Predict adds at least the same wavelets as it would add in case of separate prediction. After prediction, we have thus enlarged $\mathcal{J}$ by a single index set $\mathcal{J}' \supset \mathcal{J}$. On this new index set, we estimate the primal residual as well as the dual residual, and subsequently perform the routine Coarse. After coarsening both, the primal residual which leads to the index set $\mathcal{J}_1''$ and the dual residual which leads to the index set $\mathcal{J}_2''$, we take the union of both index sets $\mathcal{J} = \mathcal{J}_1'' \cup \mathcal{J}_2''$. Finally, the algorithm assembles and solves the system of linear equations with respect to the index set $\mathcal{J}$.

## 6.3 Laplace Equation Solved by the Single Layer Operator

Let us present our numerical results. Consider the Laplace equation (5.4), solved inside a bounded domain $\Omega$ with boundary $\Gamma$ and converted into a boundary integral equation of the form (1.3), see Chapter 1. For the following computations, we consider again the Fichera vertex $(0,1)^3 \setminus (0,0.5]^3$ as our domain. The Dirichlet data for the primal problem

are chosen as the restriction $f = u|_\Gamma$ of the polynomial $u(\boldsymbol{x}) = 4x_1^2 - 3x_2^2 - x_3^2$. The output functional under consideration is the potential evaluation

$$g(v) = \int_\Gamma \frac{v(\boldsymbol{y})}{\|\boldsymbol{x} - \boldsymbol{y}\|} \mathrm{d}\sigma_{\boldsymbol{y}} \tag{6.1}$$

in a single point $\boldsymbol{x} \in \Omega$. This potential evaluation corresponds to the application of a continuous linear functional to the density $v$. Notice that the norm of this functional obviously increases as $\boldsymbol{x} \in \Omega$ approaches the boundary $\Gamma$.

After each iteration of the adaptive algorithm, we compute an approximation $g(\boldsymbol{\rho}_\mathcal{J})$ to $g(\boldsymbol{\rho})$, given by equation (6.1), via the scalar product $\boldsymbol{g}_\mathcal{J}^t \boldsymbol{\rho}_\mathcal{J}$. We then evaluate the potential error $|u(\boldsymbol{x}) - \boldsymbol{g}_\mathcal{J}^t \boldsymbol{\rho}_\mathcal{J}|$, where $u(\boldsymbol{x})$ denotes the analytical solution.

For the comparison of Algorithm 16 and Algorithm 17, we choose the evaluation point for the functional (6.1) as $\boldsymbol{x} = (0.25, 0.25, 0.9)$. This point is located inside Fichera's vertex and close to the top boundary. Moreover, we have chosen the coarsening constant $\theta = 0.5$ and the bandwidth parameter $a = 2.5$.

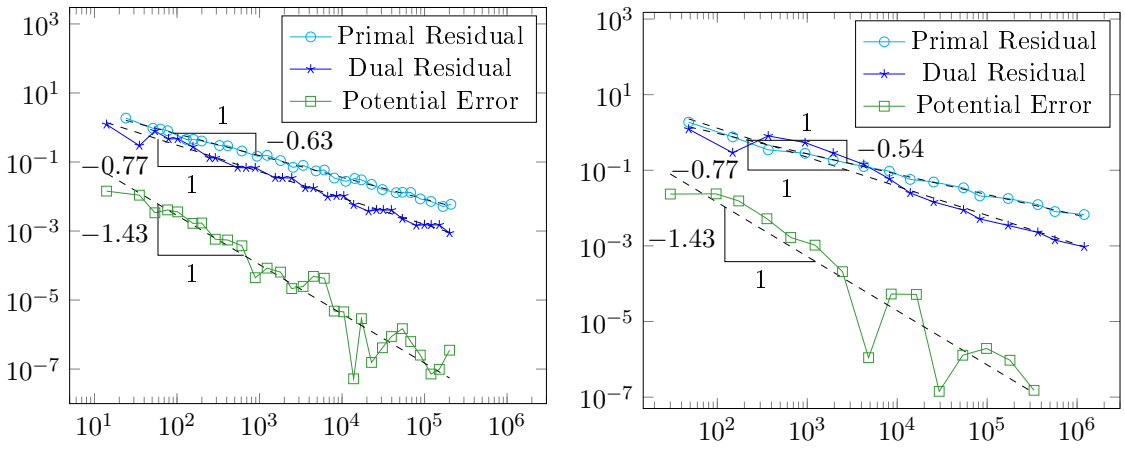



Figure 6.1: Norm of the primal residual, norm of the dual residual and the potential error for Algorithm 16 (left) and Algorithm 17 (right).

The left picture of Figure 6.1 shows the convergence histories of the primal residual, the dual residual and the potential error for Algorithm 16. In the right plot of Figure 6.1, we see the convergence histories of the primal residual, the dual residual and the potential error for Algorithm 17. We observe that the primal residual has a rate of convergence of at least $N_{\mathrm{dof}}^{-0.5}$ for both strategies, which is what we would expect, where we observe a slightly higher rate $N_{\mathrm{dof}}^{-0.63}$ of convergence for the primal residual for Algorithm 16. Also we notice that, for both algorithms, the dual residual seems to have a rate of convergence of approximately $N_{\mathrm{dof}}^{-0.75}$, which is significantly better than the rate of convergence for the primal residual. The potential error has a rate of convergence of approximately $N_{\mathrm{dof}}^{-1.4}$.

In Figure 6.2, we plot the ratios of the primal residual, the dual residual and the potential

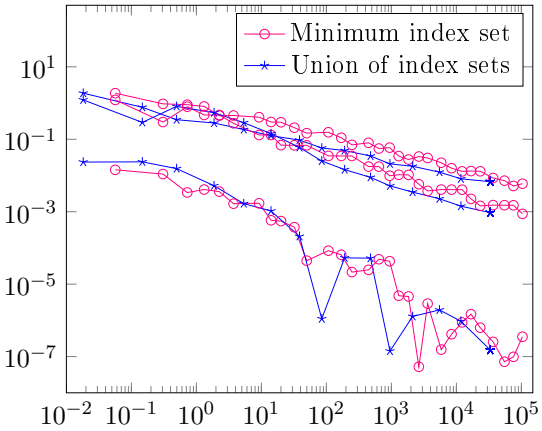



Figure 6.2: Norm of the primal residual, norm of the dual residual and potential error versus computation time for Algorithm 16 (red) and Algorithm 17 (blue).

error versus the computation time for both, Algorithm 16 and Algorithm 17. It turns out that neither algorithm is superior when it comes to computing the primal residual, the dual residual or the potential error within a given time period. Indeed, both algorithms need the same computing time to reach a desired accuracy for the quantity of interest.

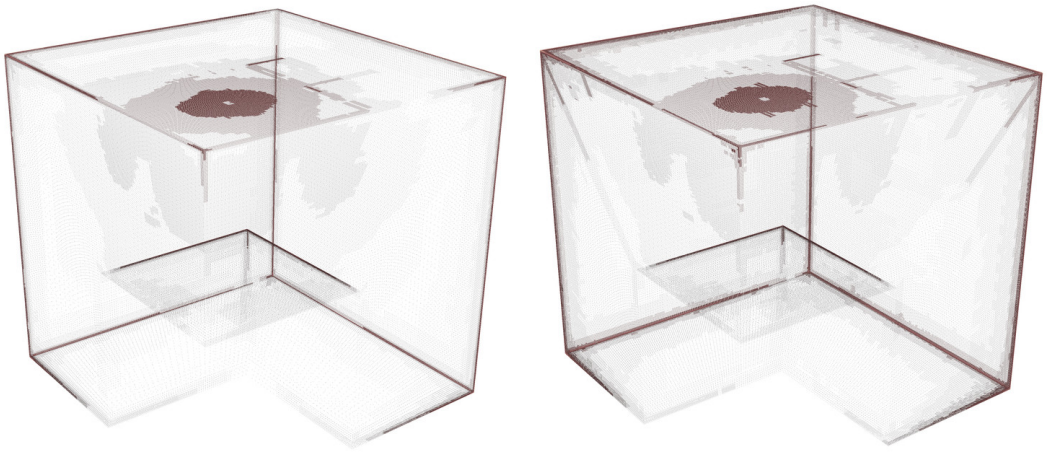


Figure 6.3: Refinement for the evaluation point $(0.25, 0.25, 0.9)$ for Algorithm 16 (left) and Algorithm 17 (right).

We should also compare the refinement which is produced by the two algorithms. In Figure 6.3, we have visualised the two refinements, where the plots are produced similarly to the plots seen in Chapter 5. We observe that the refinement for Algorithm 16 (left plot) basically coincides with the refinement for Algorithm 17 (right plot). In particular, a clear refinement is seen on the top, near from where the point $\boldsymbol{x} = (0.25, 0.25, 0.9)$ is located. Also, both algorithms refine again towards the edges of Fichera's vertex.

For the next computations, we move the evaluation more closely to the boundary, namely

we set $\boldsymbol{x} = (0.25, 0.25, 0.95)$ and perform our computations again for both strategies.

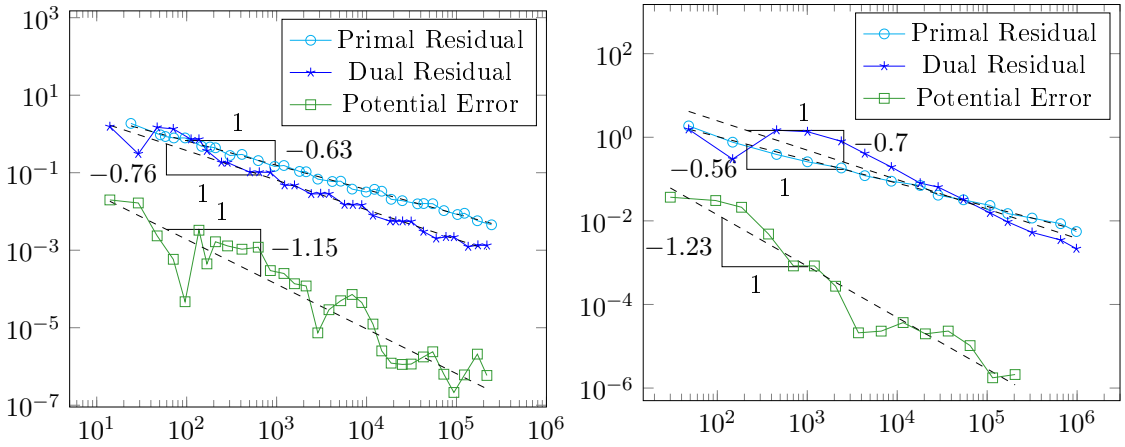



Figure 6.4: Norm of the primal residual, norm of the dual residual and potential error for Algorithm 16 (left) and Algorithm 17 (right).

In Figure 6.4, we visualise the convergence histories for the primal residual, the dual residual and the potential error versus the degrees of freedom in a log-log scale. We observe that both, the primal residual and the dual residual, show a rate of convergence of at least $N_{\mathrm{dof}}^{-0.5}$. The rate of convergence of approximately $N_{\mathrm{dof}}^{-0.7}$ for the dual residual is again higher than the rate of convergence of the primal residual. For the potential error, we observe a rate of convergence of $N_{\mathrm{dof}}^{-1.15}$ for Algorithm 16 and $N_{\mathrm{dof}}^{-1.23}$ for Algorithm 17. This is slightly less than the rate of convergence of the potential error for the evaluation point $\boldsymbol{x} = (0.25, 0.25, 0.9)$. Again, there is no significant difference between both algorithms if we compare accuracy versus computing times, see Figure 6.5.

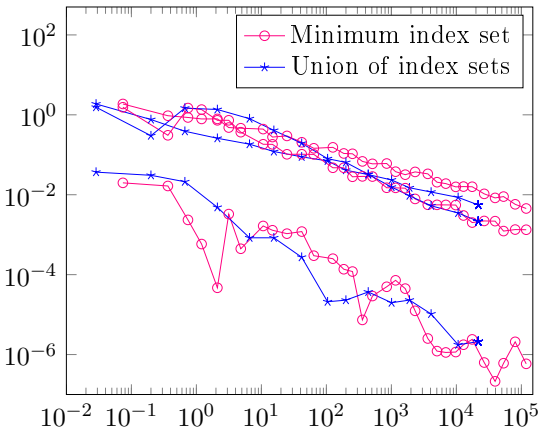



Figure 6.5: Norm of the primal residual, norm of the dual residual and potential error versus computation time for Algorithm 16 (red) and Algorithm 17 (blue).

To conclude our results for the solution of the Laplace equation by the means of the single layer potential, we visualise the refinement of both algorithms in Figure 6.6. Again, both algorithms refine towards the edges and vertices of the geometry and towards the point $\boldsymbol{x} = (0.25, 0.25, 0.95)$. If we compare the refinement on the top of the domain with

the refinement from the previous example, we notice that the refinement is slightly more localised here. Also, Algorithm 17 produces a mesh which has a much more localised refinement on the top boundary near the point $\boldsymbol{x} = (0.25, 0.25, 0.95)$.

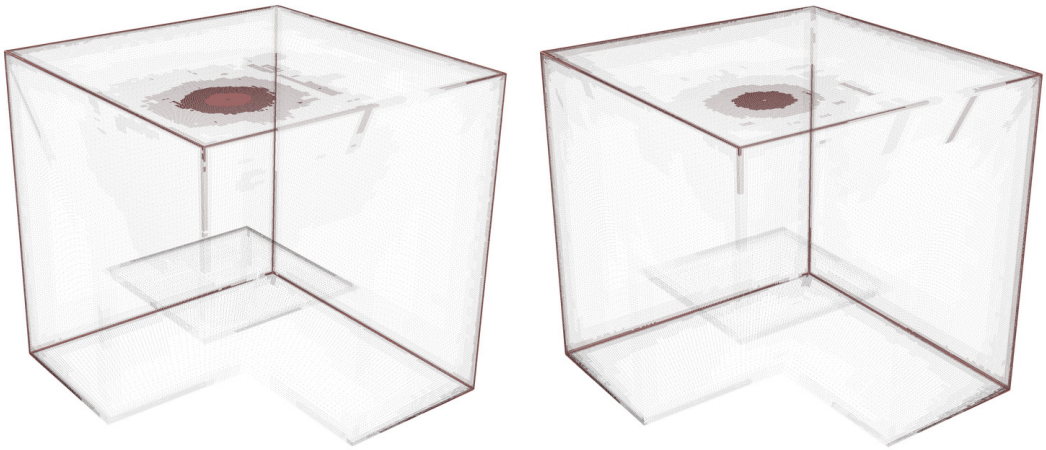


Figure 6.6: Refinement for the evaluation point $(0.25, 0.25, 0.95)$ for Algorithm 16 (left) and Algorithm 17 (right).

## 6.4 Laplace Equation Solved by the Double Layer Operator

In this section, we present an example for goal-oriented error estimation for the interior Laplace equation solved by the double layer operator. We consider the same domain as in the previous example, namely Fichera's vertex. The points $\boldsymbol{x} = (0.25, 0.25, 0.9)$ and $\boldsymbol{x} = (0.25, 0.25, 0.95)$ are again chosen as evaluation points for the potential evaluation

$$g(v) = \int_\Gamma \frac{\langle \boldsymbol{x} - \boldsymbol{y}, \boldsymbol{n}_{\boldsymbol{y}} \rangle}{\|\boldsymbol{x} - \boldsymbol{y}\|^3} v(\boldsymbol{y}) \mathrm{d}\sigma_{\boldsymbol{y}}. \tag{6.2}$$

However, the numerical results for the convergence, as well as the refinements for both points, are too similar for both Algorithms 16 and 17 as to be presented separately. Therefore, we only present our results for the point $\boldsymbol{x} = (0.25, 0.25, 0.9)$

In the left plot of Figure 6.7, we see the convergence histories of the primal residual, the dual residual and the potential error for Algorithm 16. In the right plot of Figure 6.7, we see the convergence histories of the primal residual, the dual residual and the potential error for Algorithm 17. One readily infers that the primal residual and the dual residual show the desired rate of convergence of approximately $N_{\mathrm{dof}}^{-0.5}$.

In the left plot of Figure 6.8, the refinement which is produced by Algorithm 16 is shown. In the right plot of Figure 6.8, one finds the refinement which is produced by Algorithm 17. In contrast to the previous examples, the algorithms refine differently. Algorithm 16 produces a strong local refinement on the top boundary, near to where the point $\boldsymbol{x} = (0.25, 0.25, 0.9)$ is located, whereas we observe no adaptive refinement for Algorithm 17. However, neither

algorithm refines towards the edges and corners, as it was the case for the previous example. We observe that, even though the refinement behaviour is different, the same precision with respect to the primal residual, the dual residual and the potential error for both algorithms is achieved.

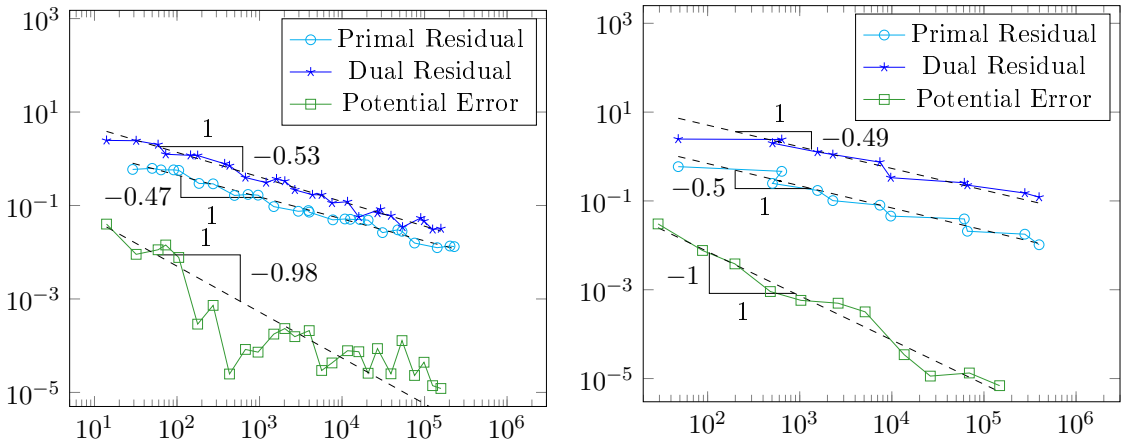



Figure 6.7: Norm of the primal residual, norm of the dual residual and potential error for Algorithm 16 (left) and Algorithm 17 (right).

Finally, we have also considered the computation times for both strategies. We observe that there is again no significant superiority of one strategy in comparison with the other. This is why we do not show another plot of the primal residual, the dual residual and the potential error versus the computation time here. Let us mention again that the same observations, concerning the primal residual, the dual residual, the potential error and the computation time, can be made if we move the point closer to the boundary, namely to $\boldsymbol{x} = (0.25, 0.25, 0.95)$.

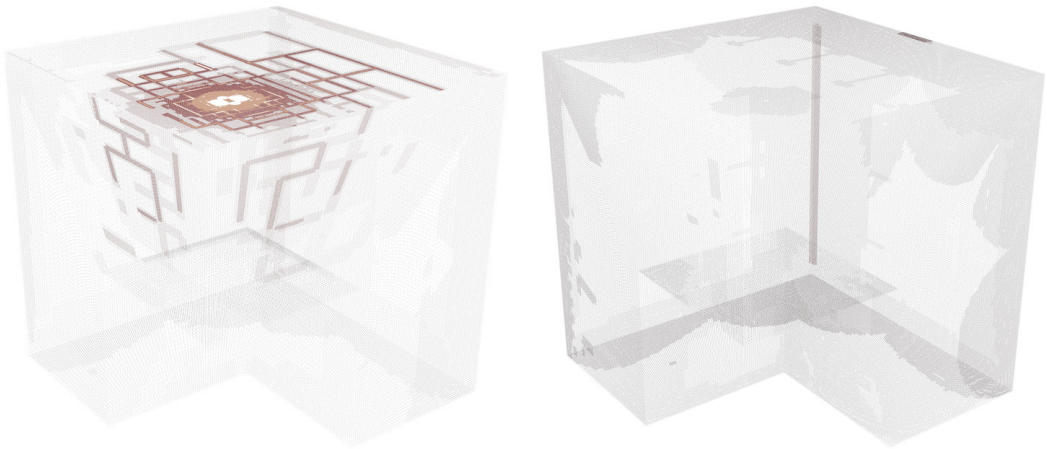


Figure 6.8: Refinement for the evaluation point $(0.25, 0.25, 0.9)$ for Algorithm 16 (left) and Algorithm 17 (right).

# Conclusion and Future Work

In this thesis, we have proposed an adaptive wavelet method for solving boundary integral equations in three dimensions by using piecewise constant wavelet functions. Especially, we introduced the boundary integral formulation of the Laplace equation via boundary potentials, as well as the boundary integral formulation for the Helmholtz equation in its direct and indirect form. In addition, we considered goal-oriented error estimation for solving the Laplace equation in a specific point inside the domain under consideration.

Throughout this thesis, we observed that several situations arise where the adaptive implementation performs especially well. One such situation is the solution of the interior Laplace equation by means of the single layer operator on a domain featuring edges and vertices, where a uniform method may no longer be feasible. Another situation is the solution of scattering problems while using a direct formulation.

If we solve the Laplace equation for a smooth right-hand side by the double layer operator, we observe the maximal rate of convergence, which, however, is also obtained by uniform refinement. The same observation can also be made when solving the Helmholtz equation by an indirect ansatz. If, however, we consider a right-hand side which admits a singularity, our adaptive wavelet algorithm performs superior to a uniform refinement, as it still produces the optimal rate of convergence by adaptively refining towards the singularity.

In the last chapter of this thesis, where we considered goal-oriented error estimation for the solution of the Laplace equation, we observed that the adaptive refinement is indeed tailored to the output functional under consideration if we consider the solution by means of the single layer operator. If we consider the solution by means of the double layer operator, we observe a different behaviour for the two presented strategies. The strategy, which takes the minimum of the two index sets produces a strong local refinement in the vicinity of where the point for the potential evaluation is located. However, for the second strategy, where we take the union of both index sets, we observe no adaptive refinement.

In conclusion, we have developed an adaptive code, which is able to compute the solution to boundary integral equations up to more than one million degrees of freedom in reasonable time. Computations in this extent, particularly for an extensive local refinement, have not been possible until now.

As the data structures for the adaptive implementation are now available, one possible continuation of this research would be the inclusion of globally continuous, piecewise bilinear wavelets. Then, the solution of boundary integral equations involving the hypersingular operator $\mathcal{W} : H^{1/2}(\Gamma) \to H^{-1/2}(\Gamma)$ could be tackled. This would, for example, include the

Burton-Miller formulation for scattering problems involving a sound-hard scatterer, which is an important example for the simulation of three-dimensional hearing, see [15].

# Bibliography


[1] W. Bangerth and R. Rannacher. *Adaptive finite element method for differential equations.* Lectures Math. ETH Zürich. Birkhäuser, Basel, 2003.

[2] M. Bebendorf. Approximation of boundary element matrices. *Numer. Math.*, 86:565–589, 2000.

[3] M. Bebendorf. *Hierarchical matrices.* Springer, Berlin-Heidelberg, 2008.

[4] M. Bebendorf and S. Rjasanow. Adaptive low-rank approximation of collocation matrices. *Computing*, 70:1–24, 2003.

[5] R. Becker, E. Estecahandy, and D. Trujillo. Weighted marking for goal-oriented adaptive finite element methods. *SIAM J. Numer. Anal.*, 49(6):2451–2469, 2011.

[6] R. Becker and R. Rannacher. An optimal control approach to a posteriori error estimation in finite element methods. *Acta Numerica 2001*, 10:1–102, 2001.

[7] G. Beylkin, R. Coifman, and V. Rokhlin. The fast wavelet transform and numerical algorithms. *Comm. Pure and Appl. Math.*, 44:141–183, 1991.

[8] P. Binev and R. DeVore. Fast computation in adaptive tree approximation. *Numer. Math.*, 97:193–217, 2004.

[9] S. Börm. *Efficient numerical methods for non-local operators: $\mathcal{H}^2$-matrix compression, algorithms and analysis*, volume 14. European Mathematical Society, Zürich, 2010.

[10] S. Börm and L. Grasedyck. Low-rank approximation of integral operators by interpolation. *Computing*, 72(3):325–332, 2004.

[11] H. Brakhage and P. Werner. Über das Dirichletsche Außenraumproblem für die Helmholtzsche Schwingungsgleichung. *Arch. Math.*, 16:325–329, 1965.

[12] J. Carnicer, W. Dahmen, and J. Peña. Local decomposition of refinable spaces. *Appl. Comput. Harm. Anal.*, 3:127–153, 1996.

[13] S.N. Chandler-Wilde and S. Langdon. A Galerkin boundary element method for high frequency scattering by convex polygons. *SIAM J. Numer. Anal.*, 45:610–640, 2007.

[14] S.N. Chandler-Wilde and P. Monk. Wave-number-explicit bounds in time-harmonic scattering. *SIAM J. Math. Anal.*, 39:1428–1455, 2008.

[15] Z.S. Chen, H. Waubke, and W. Kreuzer. A formulation of the fast multipole boundary element method (FMBEM) for acoustic radiation and scattering from three-dimensional structures. *J. Comput. Acoust.*, 16(2):303–320, 2008.

[16] A. Cohen, W. Dahmen, and R. DeVore. Adaptive wavelet methods for elliptic operator equations. Convergence rates. *Math. Comput.*, 70:27–75, 2001.

[17] A. Cohen, W. Dahmen, and R. DeVore. Adaptive wavelet methods II. Beyond the elliptic case. *Found. Comput. Math.*, 2:203–245, 2002.

[18] A. Cohen, I. Daubechies, and J.C. Feauveau. Biorthogonal bases of compactly supported wavelets. *Pure Appl. Math.*, 45:485–560, 1992.

[19] D. Colton and R. Kress. *Integral equation methods in scattering theory.* John Wiley & Sons, New York, 1983.

[20] D. Colton and R. Kress. *Inverse acoustic and electromagnetic scattering theory.* Springer, Berlin, 1992.

[21] S. Dahlke. Besov regularity for elliptic boundary value problems on polygonal domains. *Appl. Math. Lett.*, 12(6):31–36, 1999.

[22] S. Dahlke, W. Dahmen, and R.A. DeVore. Nonlinear approximation and adaptive techniques for solving elliptic operator equations. In *Multiscale wavelet methods for partial differential equations*, volume 6 of *Wavelet Analysis and Its Applications*, pages 237–283, London, 1997. Academic Press.

[23] S. Dahlke and R. DeVore. Besov regularity for elliptic boundary value problems. *Comm. Partial Differential Equations*, 22:1–16, 1997.

[24] S. Dahlke and M. Weimar. Besov regularity for operator equations on patchwise smooth manifolds. *Found. Comput. Math.*, 15:1533–1569, 2015.

[25] W. Dahmen. Decomposition of refinable spaces and applications to operator equations. *Numer. Algor.*, 5:229–245, 1993.

[26] W. Dahmen. Wavelet and multiscale methods for operator equations. *Acta Numerica*, 6:55–228, 1997.

[27] W. Dahmen, H. Harbrecht, and R. Schneider. Compression techniques for boundary integral equations. Asymptotically optimal complexity estimates. *SIAM J. Numer. Anal.*, 43(6):2251–2271, 2006.

[28] W. Dahmen, H. Harbrecht, and R. Schneider. Adaptive methods for boundary integral equations. Complexity and convergence estimates. *Math. Comput.*, 76:1243–1274, 2007.

[29] W. Dahmen and A. Kunoth. Multilevel preconditioning. *Numer. Math.*, 63:315–344, 1992.

[30] W. Dahmen, A. Kunoth, and K. Urban. Biorthogonal spline-wavelets on the interval. Stability and moment conditions. *Appl. Comput. Harmon. Anal.*, 6:132–196, 1999.

[31] W. Dahmen, A. Kunoth, and J. Vorloeper. Convergence of adaptive wavelet methods for goal-oriented error estimation. In A. Bermudez de Castro, D. Gomez, P. Quintely, and P. Salgado, editors, *Numerical mathematics and advanced applications*, pages 39–61, Berlin, 2006. Springer.

[32] W. Dahmen and R. Schneider. Composite wavelet bases for operator equations. *Math. Comput.*, 68:1533–1567, 1999.

[33] R. DeVore. Nonlinear approximation. *Acta Numerica*, 7:51–150, 1998.

[34] D.L. Donoho. Sparse components of images and optimal atomic decomposition. *Constr. Approx.*, 17:353–382, 2001.

[35] M. Duffy. Quadrature over a pyramid or cube of integrands with a singularity at the vertex. *SIAM J. Numer. Anal.*, 19:1260–1262, 1982.

[36] K. Eppler and H. Harbrecht. Second order shape optimization using wavelet BEM. *Optim. Methods Softw.*, 21:135–153, 2006.

[37] K. Eppler and H. Harbrecht. Wavelet based boundary element methods in exterior electromagnetic shaping. *Eng. Anal. Bound. Elem.*, 32:645–657, 2008.

[38] K. Eriksson, D. Estep, P. Hansbo, and C. Johnson. Introduction to adaptive methods for differential equations. *Acta Numerica*, 4:105–158, 1995.

[39] B. Faermann. Localization of the Aronszajn-Slobodeckij norm and application to adaptive boundary element methods. Part II: The three-dimensional case. *Numer. Math.*, 92(3):467–499, 2002.

[40] M. Feischl, G. Gantner, A. Haberl, D. Praetorius, and T. Führer. Adaptive boundary element methods for optimal convergence of point errors. *Numer. Math.*, 132(3):541–567, 2016.

[41] M. Feischl, M. Karkulik, J.M. Melenk, and D. Praetorius. Quasi-optimal convergence rate for an adaptive boundary element method. *SIAM J. Numer. Anal.*, 51(2):1327–1348, 2013.

[42] M. Feischl, D. Praetorius, and K.G. van der Zee. An abstract analysis of optimal goal-oriented adaptivity. *SIAM J. Numer. Anal.*, 54(3):1423–1448, 2016.

[43] M. Frazier and B. Jawerth. A discrete transform and decompositions of distribution spaces. *J. Functional Anal.*, 93:34–170, 1990.

[44] T. Gantumur. An optimal adaptive wavelet method for nonsymmetric and indefinite elliptic problems. *J. Comput. Appl. Math.*, 211(1):90–102, 2008.

[45] T. Gantumur. Adaptive boundary element methods with convergence rates. *Numer. Math.*, 124:471–516, 2013.

[46] T. Gantumur, H. Harbrecht, and R. Stevenson. An optimal adaptive wavelet method for elliptic equations without coarsening. *Math. Comput.*, 76:615–629, 2007.

[47] T. Gantumur and R. Stevenson. Computation of singular integral operators in wavelet coordinates. *Computing*, 76:77–107, 2006.

[48] G.N. Gatica, H. Harbrecht, and R. Schneider. Least squares methods for the coupling of FEM and BEM. *SIAM J. Numer. Anal.*, 41(5):1974–1995, 2003.

[49] K. Giebermann. *Schnelle Summationsverfahren zur numerischen Lösung von Integralgleichungen für Streuprobleme im* $\mathbb{R}^3$. PhD thesis, Universität Karlsruhe, Germany, 1997.

[50] L. Greengard and V. Rokhlin. A fast algorithm for particle simulation. *J. Comput. Phys.*, 73:325–348, 1987.

[51] W. Hackbusch. *Integralgleichungen – Theorie und Numerik*. B.G. Teubner, Stuttgart, 1989.

[52] W. Hackbusch. A sparse matrix arithmetic based on $\mathcal{H}$-matrices. Part I: Introduction to $\mathcal{H}$-matrices. *Computing*, 64:89–108, 1999.

[53] W. Hackbusch and Z.P. Nowak. On the fast matrix multiplication in the boundary element method by panel clustering. *Numer. Math.*, 54:463–491, 1989.

[54] H. Harbrecht. *Wavelet Galerkin schemes for the boundary element method in three dimensions*. PhD thesis, Technische Universität Chemnitz, Germany, 2001.

[55] H. Harbrecht. A Newton method for Bernoullis free boundary problem in three dimensions. *Computing*, 82:11–30, 2008.

[56] H. Harbrecht and T. Hohage. Fast methods for three-dimensional inverse obstacle scattering. *J. Integral Equations Appl.*, 19(3):237–260, 2007.

[57] H. Harbrecht, F. Paiva, C. Pérez, and R. Schneider. Biorthogonal wavelet approximation for the coupling of FEM-BEM. *Numer. Math.*, 92:325–356, 2002.

[58] H. Harbrecht, F. Paiva, C. Pérez, and R. Schneider. Wavelet preconditioning for the coupling of FEM-BEM. *Num. Lin. Alg. Appl.*, 10:197–222, 2003.

[59] H. Harbrecht and M. Peters. Comparison of fast boundary element methods on parametric surfaces. *Comput. Methods Appl. Mech. Engrg.*, 261–262:39–55, 2013.

[60] H. Harbrecht and R. Schneider. Wavelet Galerkin schemes for boundary integral equations. implementation and quadrature. *SIAM J. Sci. Comput.*, 27(4):1347–1370, 2006.

[61] H. Harbrecht, R. Schneider, and C. Schwab. Sparse second moment analysis for elliptic problems in stochastic domains. *Numer. Math.*, 109(3):167–188, 2008.

[62] M.R. Hestenes and E.L. Stiefel. Methods of conjugate gradients for solving linear systems. *J. Research Nat. Bur. Standards*, 49:409–436, 1952.

[63] G. Hsiao and W. Wendland. A finite element method for some equations of first kind. *J. Math. Anal. Appl.*, 58:449–481, 1977.

[64] T.J.R. Hughes, J.A. Cottrell, and Y. Bazilevs. Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. *Comput. Methods Appl. Mech. Engrg.*, 194(39):4135–4195, 2005.

[65] D. Huybrechs, J. Simoens, and S. Vandevalle. A note on wave number dependence of wavelet matrix comression for integral equations with oscillatory kernel. *J. Comput. Appl. Math.*, 172:233–246, 2004.

[66] S. Jaffard. Wavelet methods for fast resolution of elliptic equations. *SIAM J. Numer. Anal.*, 29:965–986, 1992.

[67] H. Johnen and K. Scherer. On the equivalence of the $K$-functional and moduli of continuity and some applications. In W. Schempp and K. Zeller, editors, *Constructive theory of functions of several variables (Proc. Conf., Math. Res. Inst., Oberwolfach, 1976)*, volume 571, pages 119–140, Berlin, 1977. Springer.

[68] A. Kirsch. *Generalized boundary value and control problems for the Helmholtz equations.* Habilitationsschrift, Universität Göttingen, 1984.

[69] D.E. Knuth. *Sorting and Searching*, volume 3 of *The art of computer programming.* Addison-Wesley, Redwood City, USA, 1998.

[70] R. Kress. Minimizing the condition number of boundary integral operators in acoustic and electromagnetic scattering. *Q. J. Mech. Appl. Math.*, 38:323–341, 1985.

[71] R. Kress. *Linear Integral Equations.* Springer, Berlin-Heidelberg, 1989.

[72] R. Leis. Zur dirichletschen Randwertaufgabe des Außenraumes der Schwingungsgleichung. *Mathematische Zeitschr.*, 90:203–211, 1965.

[73] W. Magnus. Über Eindeutigkeitsfragen bei einer Randwertaufgabe von $\delta u + ku = 0$. *Jahresber. Dtsch. Math. Ver.*, 52:177–188, 1943.

[74] M. Maischak and E.P. Stephan. The *hp*-version of the boundary element method in $\mathbf{R}^3$: the basic approximation results. *Math. Methods Appl. Sci.*, 20(5):461–476, 1997.

[75] M. Mitrea. Boundary value problems and hardy spaces associated to the Helmholtz equation in Lipschitz domains. *J. Math. Anal. Appl.*, 202:819–842, 1996.

[76] M.S. Mommer and R.P. Stevenson. A goal-oriented adaptive finite element method with convergence rates. *SIAM J. Numer. Anal.*, 47(2):861–886, 2009.

[77] O.I. Panich. On the question of the solvability of the exterior boundary-value problems for the wave equation and Maxwells equation. *Russian Math. Surveys*, 20:221–226, 1965.

[78] F. Rellich. über das asymptotische Verhalten der Lösungen von $\delta u + \lambda u = 0$ in unendlichen Gebieten. *Jahresber. Dtsch. Math. Ver.*, 53:57–65, 1943.

[79] V. Rokhlin. Rapid solution of integral equations of classical potential theory. *J. Comput. Phys.*, 60:187–207, 1985.

[80] Y. Saad and M.H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 7(3):856–869, 1986.

[81] S. Sauter and C. Schwab. Quadrature for the *hp*-Galerkin BEM in $\mathbb{R}^3$. *Numer. Math.*, 78:211–258, 1997.

[82] S. Sauter and C. Schwab. *Randelementmethoden: Analyse, Numerik und Implementierung schneller Algorithmen.* B.G. Teubner, Wiesbaden, 2004.

[83] S. Sauter and C. Schwab. *Boundary element methods.* Springer, Berlin, 2011.

[84] R. Schneider. *Multiskalen- und Wavelet-Matrixkompression: Analysisbasierte Methoden zur effizienten Lösung großer vollbesetzter Gleichungssysteme.* B.G. Teubner, Stuttgart, 1998.

[85] A. Sommerfeld. Die Greensche Funktion der Schwingungsgleichung. *Jahresber. Dtsch. Math. Ver.*, 21:309–353, 1912.

[86] O. Steinbach. *Numerische Näherungsverfahren für elliptische Randwertprobleme: Finite Elemente und Randelemente.* B.G. Teubner, Stuttgart-Leipzig-Wiesbaden, 2003.

[87] R. Stevenson. On the compressibility of operators in wavelet coordinates. *SIAM J. Math. Anal.*, 35(5):1110–1132, 2004.

[88] E.E. Tyrtyshnikov. Mosaic sceleton approximation. *Calcolo*, 33:47–57, 1996.

[89] T. von Petersdorff and C. Schwab. Wavelet approximation for first kind integral equations on polygons. *Numer. Math.*, 74:479–519, 1996.

[90] T. von Petersdorff and C. Schwab. Fully discrete multiscale Galerkin BEM. In *Multiscale Wavelet Methods for PDEs*, pages 287–346. Academic Press, San Diego, 1997.

[91] T. von Petersdorff and C. Schwab. Sparse finite element methods for operator equations with stochastic data. *Appl. Math*, 51:145–180, 2006.

[92] C.H. Wilcox. A generalization of theorems of Rellich and Atkinson. *Proc. Amer. Math. Soc.*, 7:271–276, 1956.