JOHANNES KEPLER
UNIVERSITÄT LINZ
Netzwerk für Forschung, Lehre und Praxis

Boundary Element Solvers for Linear Water Wave Simulation in a Model Basin

Diplomarbeit

zur Erlangung des akademischen Grades

Diplomingenieur

in der Studienrichtung

Technische Mathematik

Angefertigt am *Institut für Numerische Mathematik*

Betreuung:

O.Univ.Prof. Dipl.-Ing. Dr. Ulrich Langer

Eingereicht von:

Clemens Hofreither

Linz, September 2008

Abstract

We derive a three-dimensional, linearized model for the time-dependent simulation of free-surface water waves in a test basin. For the numerical solution of this problem, we apply a Runge-Kutta scheme for time discretization and show that the problem to be solved at each time step corresponds to the evaluation of a Dirichlet-to-Neumann map on the free surface of the domain. We use the Galerkin Boundary Element Method for the approximate evaluation of this operator. Data-sparse matrix approximation methods based on hierarchical matrix representation are used in order to solve the resulting large, dense linear systems in a quasi-optimal manner. We perform numerical tests and report results on runtime and accuracy of such an algorithm.

Contents

Chapter 1

Introduction

1.1 Motivation

In hydrodynamic laboratories, new designs for maritime vessels like freight carriers or ferries are first tested at a model scale in a controlled environment. In order to be able to test ships under realistic wave conditions, the basins in which these tests are performed are typically equipped with so-called wave makers along one or more of their sides; these are arrays of independently movable elements which may be controlled in such a way as to excite waves at the surface of the water.

Clearly, there are different profiles of waves under which different kinds of ships should be tested depending on the environmental factors under which they are planned to operate in the real world. It is therefore crucial to be able to create a range of such wave profiles in a test basin by means of the wave makers. It is not trivial to determine the movement of the wave makers required to create a given wave profile, nor, inversely, the wave profile which ensues from a certain wave maker configuration. It is therefore very useful to have a numerical simulation of the wave behaviour in such a basin for a predetermined wave maker motion. Using such a simulation, a human operator may iteratively tune the wave maker parameters in order to create a desired wave profile without taking up valuable testing time in an actual wave basin. Conceivably, such a simulation could also be used in order to define an objective function for an optimization problem in order to fully automatize the process.

Ultimately, one might want to develop so-called Numerical Wave Tanks (NWTs): here, the goal is no longer the simulation of existing wave basins, but the complete replacement of these facilities by numerical simulation. To achieve this, more detailed models which also include wave-body interaction for the vessel being tested are needed. These developments are however beyond the scope of this work, and we will here focus on the simulation of wave generation without paying attention to any interference by bodies

floating in the test basin. On the subject of NWTs, we direct the interested reader to the references in [13].

1.2 Review of previous work

There is a wealth of literature on the topic of simulation of both linear and nonlinear waves in settings comparable to the one employed herein. We will only cite a select few which were used for reference during the creation of this work.

A thorough discussion of various forms of wave phenomena may be found in [14]; in particular, this book was consulted for the derivation of the governing equations for the wave basin.

[13] is a quite comprehensive treatment of both the modelling and the numerical solution of nonlinear waves by the use of a Finite Element Method.

In [12], a discontinuous Galerkin Finite Element Method is developed to solve the same linearized wave problem formulation that we use here, albeit in a two-dimensional setting.

In [9], a *hp*/spectral element method for the solution of both linear and nonlinear free-surface flow is described.

[5] describes a Lagrangian approach using a FEM for the fully nonlinear wave model where the dynamic body of water is transformed onto a fixed domain.

It should also be noted that this work is a continuation of a project finished for a project seminar at the Institute of Computational Mathematics of the Johannes Kepler University Linz in the summer semester 2007. In that project, a similar problem to the one described herein was solved, but restricted to two dimensions.

1.3 Overview of the thesis

This work is for the most part concerned with the instationary solution of the linearized potential formulation of inviscid, incompressible free-surface flow in three dimensions. The major distinguishing factor to the works cited above is the use of a Galerkin Boundary Element Method for the evaluation of the Dirichlet-to-Neumann map which arises in every time step. Boundary Element Methods have long been known as an attractive way to solve such problems in two-dimensional settings. However, the requirement of solving linear systems with large dense matrices has long inhibited their use in the three-dimensional case. Only relatively recently, by the development of so-called data-sparse approximation techniques, has their use become viable in these cases. (See [2] and the references therein.)

Chapter 2 presents the derivation of the linearized equations which govern the flow of a body of water in a model basin.

In Chapter 3, we introduce the concepts of the Boundary Element Method and derive a specific symmetric formulation of a Galerkin BEM which will be used in the numerical solution of the wave problem.

Chapter 4 describes the techniques needed to approximate the BEM system matrices in a data-sparse way in order to obtain acceptable run-time complexity and memory usage.

Chapter 5 presents numerical tests for model boundary value problems in order to determine the performance of the BEM, as well as for the instationary wave problem iteration itself. Where possible, we give error behaviour measurements, and we time the execution of the algorithm in dependence of the grid size.

Chapter 6 summarizes the findings of the thesis and sketches possible further developments.

Chapter 2

Mathematical Model of the Test Basin

Let us consider a three-dimensional, simply connected, time-dependent body of water, $\Omega(t) \subset \mathbb{R}^3$, which is oriented such that gravity applies in negative z direction; in other words, the unit z vector is considered to point "upwards". Some parts of the boundary of $\Omega(t)$ represent the walls of the basin and are therefore stationary. (We neglect for the moment the effect of the moving wave maker elements.) We shall call these parts of the boundary the *constrained surface.* Other parts of the boundary represent the area where the body of water interfaces with the air above it. These parts will generally be in motion, and we call them the *free surface* of $\Omega(t)$.

Together, the constrained and the free surface make up the entirety of the boundary $\Gamma(t) = \partial\Omega(t)$ of our domain. The constrained surface is given, the free surface is *a priori* unknown. Determining the location of the free surface as a function of time is our main objective.

This chapter deals with the derivation of the model problem which will be used to simulate the motion of the free surface. First, in section 2.1, we derive general formulae governing the local behaviour of such a body of water. In section 2.2, we derive appropriate boundary conditions for both the constrained and the free surface. In section 2.3 we seek to simplify the resulting model by linearizing it. (More in-depth discussion of the subject of these three chapters may be found in [14].) Section 2.4 is concerned with the time discretization of this linearized model. Finally, in section 2.5, we make some preliminary observations about the evaluation of the Dirichlet-to-Neumann map which arises during time discretization.

2.1 Continuum mechanical description of a body of water

We start our derivation from the well-known equations for an ideal, incompressible fluid,

$$\frac{Dv}{Dt} = -\frac{1}{\rho}\nabla p - ge_z, \tag{2.1}$$

$$\nabla \cdot v = 0, \tag{2.2}$$

where we use the symbols

$v(x, y, z, t)$ ... velocity (vector-valued function),
$p(x, y, z, t)$ ... pressure (scalar-valued function),
ρ ..... density (constant),
g ..... gravitational acceleration (constant),
e_z ..... unit vector in z-direction

as well as ∇ for the gradient, $\nabla\cdot$ for the divergence and $\frac{D}{Dt}$ for the material derivative of a quantity. (For reference, see e.g. [6]).

Recalling that, by definition, the material derivative has the form

$$\frac{Df}{Dt} = \frac{\partial f}{\partial t} + (v \cdot \nabla)f$$

and using the vector identity

$$(v \cdot \nabla)v = (\nabla \times v) \times v + \nabla(\frac{1}{2}v^2)$$

(which is easily proved by elementary calculation) we get from (2.1)

$$\frac{\partial v}{\partial t} + \omega \times v + \nabla(\frac{1}{2}v^2) = -\frac{1}{\rho}\nabla p - ge_z, \tag{2.3}$$

where we have introduced as a new symbol the vorticity $\omega = \nabla \times v$. In our application, it is reasonable to assume irrotational flow, that is $\omega = 0$, and (2.3) simplifies to

$$\frac{\partial v}{\partial t} + \nabla(\frac{1}{2}v^2) = -\frac{1}{\rho}\nabla p - ge_z. \tag{2.4}$$

Since we have assumed $\Omega(t)$ to be simply connected, the flow v is irrotational if and only if v is conservative. Hence we may introduce a potential $v = \nabla\phi$ and obtain, by integration,

$$\frac{\partial \phi}{\partial t} + \frac{1}{2}(\nabla\phi)^2 = \frac{p_0 - p}{\rho} - gz. \tag{2.5}$$

Here we have introduced an integration constant in the form of a constant reference pressure $p_0 \in \mathbb{R}$ which will be used in section 2.2 to derive boundary conditions at the free surface.

By a simple rewriting of (2.5) and using the incompressibility condition (2.2), we obtain the basic set of equations for our problem:

$$\triangle\phi = 0 \tag{2.6}$$

$$\frac{p - p_0}{\rho} = -\frac{\partial\phi}{\partial t} - \frac{1}{2}(\nabla\phi)^2 - gz \tag{2.7}$$

2.2 Boundary conditions

For our model, we assume that $\Omega(t)$ is aligned such that the free surface, when at rest, aligns with the plane $z = 0$. We denote this resting free surface by

$$\widetilde{\Gamma_F} \subset \mathbb{R}^2 \times \{0\}.$$

To describe the time-dependent perturbations of the free surface, we choose a simple elevation-based parameterization. We introduce a perturbation function $\zeta : (x, y, t) \mapsto \mathbb{R}$ and assume that the free surface $\Gamma_F(t) \subset \partial\Omega(t)$ is given by

$$\Gamma_F(t) = \left\{(x, y, z) \in \mathbb{R}^3 : (x, y, 0) \in \widetilde{\Gamma_F}, z = \zeta(x, y, t)\right\}.$$

See fig. 2.1 for a two-dimensional sketch.

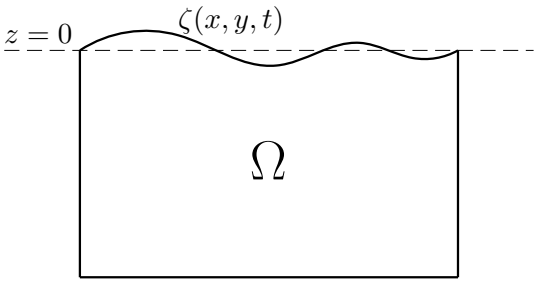

Figure 2.1: A sketch of the free surface parametrization ζ.

Note that this parameterization is not fully general. In particular, in this model it is impossible for the crests of waves to travel over the top of troughs lying ahead of them, thus forming so-called breaking waves. Refer to fig. 2.2 for a sketch of such a situation. A model general enough to handle these situations would however complicate matters significantly, and for the sake of simplicity we will stick to the more basic representation.

The constrained surface (which is known in advance) is denoted by Γ_C. As noted in the introduction to this chapter, we have

$$\partial\Omega(t) = \Gamma(t) = \Gamma_F(t) \,\dot\cup\, \Gamma_C.$$

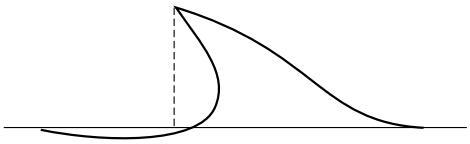

Figure 2.2: Sketch of "breaking wave" which cannot be represented in simple elevation-based parameterization.

We now have to consider which boundary conditions to impose on the different parts of the boundary. On the constrained surface, i. e. the walls of the basin, we assume that there is a prescribed normal velocity $g_N : (x, y, z, t) \mapsto \mathbb{R}$ which translates via

$$g_N = v \cdot n = \nabla\phi \cdot n \quad \text{on } \Gamma_C$$

into a standard Neumann boundary condition for ϕ at any given time t. In particular, for those parts of the constrained surface which represent fixed walls, we will have $g_N \equiv 0$. On the other hand, the parts of the basin wall which are equipped with wave maker elements may be modelled by varying the normal velocity at those boundary segments periodically.

The more complicated boundary conditions are of course those on the free surface. In fact, because of the time-dependence of this surface, we will require two boundary conditions here instead of just one. These are known as the *kinematic* and the *dynamic* boundary condition.

2.2.1 The kinematic boundary condition

The free surface $\Gamma_F(t)$ may be characterized by the fact that particles which are on the surface stay on the surface. In other words, for all $(x, y, z) \in \Gamma_F(t)$, we enforce

$$\frac{D}{Dt}(\zeta(x, y, t) - z) = 0,$$

which may be rearranged successively to yield

$$\begin{aligned} \frac{D}{Dt}\zeta(x, y, t) &= \frac{D}{Dt}z, \\ \frac{\partial\zeta}{\partial t} + v \cdot \nabla\zeta &= v \cdot e_z, \\ \frac{\partial\zeta}{\partial t} + \nabla\phi \cdot \nabla\zeta &= \frac{\partial\phi}{\partial z}. \end{aligned} \tag{2.8}$$

(2.8) is the kinematic boundary condition for our free surface.

2.2.2 The dynamic boundary condition

From physics, we get that the pressure in the water and the pressure in the air must be equal at the free surface. Assuming pressure in the air is

$p_0 = \text{const.}$, which is a reasonable simplification, we get

$$p(x, y, z, t) = p_0 \quad \text{on } \Gamma_F(t).$$

Recalling (2.7), this yields the dynamic boundary condition

$$\frac{\partial \phi}{\partial t} + \frac{1}{2}(\nabla \phi)^2 + gz = 0 \quad \text{on } \Gamma_F(t). \tag{2.9}$$

2.3 Linearization

For all $(x, y, z) \in \Gamma_F(t)$, we have derived the boundary conditions

$$\begin{aligned} \frac{\partial \zeta}{\partial t} + \nabla \phi \cdot \nabla \zeta &= \frac{\partial \phi}{\partial z}, \\ \frac{\partial \phi}{\partial t} + \frac{1}{2}(\nabla \phi)^2 + gz &= 0. \end{aligned}$$

Assuming that velocities $\nabla \phi$ and perturbations ζ are small, we may linearize by omitting their products to obtain

$$\begin{aligned} \frac{\partial \zeta}{\partial t} &= \frac{\partial \phi}{\partial z}, \\ \frac{\partial \phi}{\partial t} &= -gz. \end{aligned}$$

Recalling that $(x, y, z) \in \Gamma_F(t) \implies z = \zeta(x, y, t)$, we may equivalently write

$$\begin{aligned} \frac{\partial \zeta}{\partial t} &= \frac{\partial \phi}{\partial z}, \\ \frac{\partial \phi}{\partial t} &= -g\zeta. \end{aligned}$$

Finally, noting that ζ was assumed to be small, we may conclude that

$$\Gamma_F(t) \approx \widetilde{\Gamma_F}.$$

In other words, for small perturbations, the time-dependent free surface is well approximated by the free surface at rest. A further step of linearization may therefore be taken by applying these conditions on $z = 0$ rather than on $z = \zeta(x, y, t)$, yielding

$$\begin{aligned} \frac{\partial \zeta}{\partial t} &= \frac{\partial \phi}{\partial n} \qquad \text{on } \widetilde{\Gamma_F}, \\ \frac{\partial \phi}{\partial t} &= -g\zeta \qquad \text{on } \widetilde{\Gamma_F} \end{aligned}$$

where n denotes the outward unit normal vector to the linearized surface.

In other words, we approximate the time-dependent domain $\Omega(t)$ by the stationary domain Ω with the boundary $\partial\Omega = \Gamma = \widetilde{\Gamma_F} \cup \Gamma_C$. The sought-after wave profile is thus no longer encoded in the geometry of our computational domain, but only in the perturbation function ζ. This will become a great benefit later on when numerical methods are applied to the problem, as we will then be able to use the same space discretization for every time step instead of using a different discretization in each time step.

To summarize, we have now derived the fully linearized model problem

$$\left.\begin{aligned} \triangle\phi &= 0 && \text{in } \Omega, \\ \frac{\partial\phi}{\partial n} &= g_N && \text{on } \Gamma_C, \\ \frac{\partial\zeta}{\partial t} &= \frac{\partial\phi}{\partial n} && \text{on } z = 0, \\ \frac{\partial\phi}{\partial t} &= -g\zeta && \text{on } z = 0. \end{aligned}\right\} \tag{2.10}$$

2.4 Time discretization

It is not immediately obvious how (2.10) can be efficiently solved. Let us first note that the time dependence enters only at the (linearized) free surface, $z = 0$. Here we have functions $\zeta(x, y, t)$ and $\phi(x, y, 0, t) =: \phi_D(x, y, t)$ whose time derivatives are described in dependence of the respective other function; however, $\frac{\partial\zeta}{\partial t}$ depends not on the values of ϕ_D itself, but on the normal derivative $\frac{\partial\phi}{\partial n}$ of the potential at the free surface.

In order to make this structure more clear, we now attempt a rewriting of the problem in the form of an Ordinary Differential Equation (ODE). To this end, we introduce the Dirichlet-to-Neumann map or Steklov-Poincaré operator $\mathcal{S}(t)$ which maps some Dirichlet data g_D on $\widetilde{\Gamma_F}$ to the Neumann data as follows:

$$\mathcal{S}(t) : g_D \mapsto \left.\frac{\partial u}{\partial n}\right|_{\widetilde{\Gamma_F}}$$

where $u : \Omega \to \mathbb{R}$ such that

$$\left.\begin{aligned} \triangle u &= 0 && \text{in } \Omega, \\ \frac{\partial u}{\partial n} &= g_N(t) && \text{on } \Gamma_C, \\ u &= g_D && \text{on } \widetilde{\Gamma_F}. \end{aligned}\right\} \tag{2.11}$$

Using this operator, the model problem may now be rewritten in the form of a system of two coupled ODEs,

$$\frac{\partial}{\partial t}\begin{pmatrix}\phi_D \\ \zeta\end{pmatrix} = \begin{pmatrix}0 & -g \\ \mathcal{S}(t) & 0\end{pmatrix}\begin{pmatrix}\phi_D \\ \zeta\end{pmatrix}. \tag{2.12}$$

We note that $\mathcal{S}(t)$ is in general a nonlinear operator, thus making the ODE system (2.12) nonlinear as well.

From the basic theory of ODEs, it should now be clear that in addition to the conditions specified so far, we will also require initial conditions for ζ and ϕ_D. We will simply use constant zero functions for both initial values. In the context of our model problem, this means that the free surface is non-disturbed and at rest initially.

For the solution of this ODE system in a time interval $[0, T]$, we introduce a discretization of the time axis $0 = t_0 < t_1 < \ldots < t_N = T$ and the notations

$$\begin{aligned} \phi_D^{(n)}(x, y) &:= \phi_D(x, y, t_n), \\ \zeta^{(n)}(x, y) &:= \zeta(x, y, t_n), \\ \tau^{(n)} &:= t_{n+1} - t_n. \end{aligned}$$

The finite series of functions $(\phi_D^{(n)})_{n=0,\ldots,N}$ and $(\zeta^{(n)})_{n=0,\ldots,N}$ are thus time-discretized versions of our free surface functions ϕ_D and ζ, respectively.

We then apply a classical explicit fourth-order Runge-Kutta method, which has the following Butcher tableau:

$$\begin{array}{c|cccc} 0 & & & & \\ \frac{1}{2} & \frac{1}{2} & & & \\ \frac{1}{2} & 0 & \frac{1}{2} & & \\ 1 & 0 & 0 & 1 & \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array}$$

The resulting method, when written out in full, may be seen in fig. 2.3 on page 14. Note that by suitable dimensional transformations, we may assume $g = 1$.

2.5 Evaluating the Dirichlet-to-Neumann map

Examining fig. 2.3, the only part missing in order to be able to actually solve our problem is a method to evaluate the Dirichlet-to-Neumann operator $\mathcal{S}(t)$. However, upon inspection of its definition (see (2.11)), it becomes clear that evaluation of $\mathcal{S}(t)$ is at its core equivalent to the solution of Laplace's equation $\triangle\phi = 0$ in Ω with mixed Dirichlet and Neumann boundary conditions. This is a standard problem in the theory of partial differential equations and quite thoroughly studied. For its numerical solution, several well-established methods are available. Certainly the most widely-used of these methods is the Finite Element Method (FEM). One alternative is the Finite Volume Method (FVM), which is especially popular in the computational fluid dynamics community. For very simple geometry as we will employ later in our numerical example, even a simple Finite Difference Method (FDM) would be applicable.

$$
\begin{aligned}
S_1 &= \mathcal{S}(t_n)(\phi_D^{(n)}) \\
g_{1,\phi_D} &= \phi_D^{(n)} - \frac{\tau^{(n)}}{2}\zeta^{(n)} \\
g_{1,\zeta} &= \zeta^{(n)} + \frac{\tau^{(n)}}{2}S_1
\end{aligned}
$$

$$
\begin{aligned}
S_2 &= \mathcal{S}(t_n + \frac{\tau^{(n)}}{2})(g_{1,\phi_D}) \\
g_{2,\phi_D} &= \phi_D^{(n)} - \frac{\tau^{(n)}}{2}g_{1,\zeta} \\
g_{2,\zeta} &= \zeta^{(n)} + \frac{\tau^{(n)}}{2}S_2
\end{aligned}
$$

$$
\begin{aligned}
S_3 &= \mathcal{S}(t_n + \frac{\tau^{(n)}}{2})(g_{2,\phi_D}) \\
g_{3,\phi_D} &= \phi_D^{(n)} - \tau g_{2,\zeta} \\
g_{3,\zeta} &= \zeta^{(n)} + \tau S_3
\end{aligned}
$$

$$
\begin{aligned}
S_4 &= \mathcal{S}(t_n + \tau)(g_{3,\phi_D}) \\
\phi_D^{(n+1)} &= \phi_D^{(n)} - \frac{\tau^{(n)}}{6}\left(\zeta^{(n)} + 2g_{1,\zeta} + 2g_{2,\zeta} + g_{3,\zeta}\right) \\
\zeta^{(n+1)} &= \zeta^{(n)} + \frac{\tau^{(n)}}{6}\left(S_1 + 2S_2 + 2S_3 + S_4\right)
\end{aligned}
$$

Figure 2.3: The full explicit fourth-order Runge-Kutta time integration scheme for (2.12).

All these methods share the property that the entire volume Ω has to be discretized into a grid of some form. Likewise, they all produce as their solution the values of the potential ϕ over the entire grid. Looking again at (2.11), we see that our requirements are different: we actually need the normal velocities $\frac{\partial \phi}{\partial n}$, and we only need them on the Dirichlet boundary. In particular, we do not require the calculation of the values of ϕ at any point within Ω. For the afore-mentioned methods, there are ways to recover the normal velocities given the potential field ϕ; a naive approach however incurs a loss of accuracy, thus creating a need for more sophisticated, nontrivial velocity recovery techniques.

It might therefore be advantageous to employ a method which operates directly on the boundary values of ϕ and $\frac{\partial \phi}{\partial n}$. Such a method, namely the Boundary Element Method (BEM), is in fact available. The next chapter is concerned with introducing the general concepts behind this method and with developing a particular instance of the BEM which will be used to solve the problem at hand.

Chapter 3

The Boundary Element Method

3.1 Motivation and overview

Given is an elliptic differential equation in some domain Ω,

$$(Lu)(x) = f(x) \quad \forall x \in \Omega,$$

with appropriate boundary conditions of the first, second and/or third kind. It is our goal to determine u in dependence of the right-hand side f and the boundary conditions. The basic idea of the Boundary Element Method (BEM) is to reformulate this problem as an integral equation on the boundary $\Gamma = \partial\Omega$ of the domain.

The main tools to accomplish this are the third Green's identity, also known as the *representation formula*, and the *fundamental solution* of the elliptic differential operator L. By the use of these tools, we derive an identity for the *Cauchy data*, $u|_\Gamma$ and $\frac{\partial u}{\partial n}|_\Gamma$, on the boundary. By inserting the known boundary conditions for the corresponding parts of the Cauchy data, we obtain an integral equation over Γ which can then be solved using any of the known techniques from the study of such problems.

In this way, we can calculate the missing parts of the Cauchy data. If the solution is required in the entire domain Ω, it may be obtained by means of the representation formula.

The advantages of the BEM over a standard Finite Element Method (FEM) may be summarized as follows:

- By only considering the boundary, the dimension of the problem is reduced by one. For instance, a two-dimensional problem will be reduced to a one-dimensional one over the curve representing the boundary of the domain. We will however see that this advantage is counterbalanced by other factors.

- Similarly, instead of discretizing the entire domain, we only have to discretize its lower-dimensional boundary, which is an immediate simplification during the meshing phase.

- In some applications, the Cauchy data is really what one is interested in. In those cases, the BEM has a certain advantage over a FEM, especially if the normal derivative $\frac{\partial u}{\partial n}|_\Gamma$ is of interest: with a FEM, this quantity has to be recovered in a nontrivial way from the known values of u in order to avoid a loss of accuracy. This advantage applies to the problem we have established in Chapter 2.

- If required, both the solution and its derivative can be obtained with the same accuracy in the entire domain Ω by means of the representation formula.

- Both inner and outer boundary value problems (i. e. those in the complement of some bounded domain Ω) can be treated using the same general technique.

Naturally, the BEM also has certain disadvantages as compared to the FEM:

- As hinted at above, the BEM relies not only on the existence, but also the explicit knowledge of the fundamental solution of the differential operator defining the boundary value problem. This immediately precludes problems where the fundamental solution is not known from being treated with such a method.

- The integral kernels arising in the formulation of such boundary integral equations are singular. While this is certainly an advantage with respect to unique solvability of the resulting integral equations, it leads to complications when generating the system matrices numerically.

- There are certain theoretical and practical difficulties associated with the treatment of singularities on edges and corners of the domain, especially for the collocation boundary element technique.

- As we shall see, the system matrices arising from discretizing boundary integral equations are dense, not sparse as in common FEM approaches. This greatly offsets the above-mentioned advantage of dimensional reduction and renders the treatment of three- or higher-dimensional problems using a naive approach nearly intractable. This particular disadvantage can be remedied by the use of data-sparse approximations of the dense BEM matrices; we will elaborate on this in Chapter 4.

In the following, we will expand on the concepts mentioned in this introduction. Section 3.2 describes the transformation of the boundary value problem into an infinite-dimensional boundary integral equation. Section 3.3 then introduces a Galerkin approximation for this integral equation. Section 3.4 is concerned with the details of computing the BEM matrix entries. Finally, in section 3.5, we expend some notes on the properties of BEM matrices and on how to solve the resulting linear system.

3.2 Boundary integral equations

From now on, in accordance with problem (2.10), we will restrict ourselves to the three-dimensional case; the two-dimensional derivation is largely analogous. Furthermore, we assume that $\Omega \subset \mathbb{R}^3$ is a bounded, simply connected Lipschitz domain with boundary $\Gamma = \partial\Omega \in C^{0,1}$ and outward unit normal vector n. As mentioned in section 3.1, outer boundary value problems on $\overline{\Omega}^{\mathsf{C}}$ could be treated in much the same fashion, but again we stick to the relevant problem at hand. A comprehensive treatment of both interior and exterior problems for various differential operators using the BEM can be found in [8].

3.2.1 Fundamental solutions

We shall first introduce the notion of the fundamental solution of a differential operator.

Definition 3.2.1. Let $L \equiv L_x$ be a differential operator. $E(x, y)$ is called *fundamental solution* of L if and only if for all $x, y \in \mathbb{R}^3$,

$$L_x E(x, y) = \delta(x - y) \qquad \text{in } \mathcal{D}'(\Omega). \tag{3.1}$$

The right-hand side δ here refers to the Dirac delta distribution, and therefore this relation is to be satisfied in the distributional space $\mathcal{D}'(\Omega)$. Let us briefly reiterate the precise meaning of this identity.

For bounded domains Ω, we can set $\mathcal{D}(\Omega) = C_0^\infty(\Omega)$, that is, the space of all infinitely continuously differentiable functions with compact support in Ω. For a sequence $(\phi_n)_{n\in\mathbb{N}}$ in $C_0^\infty(\Omega)$ we define convergence by

$$\begin{aligned}\phi_n \to 0 \text{ in } C_0^\infty(\Omega) \Leftrightarrow\ & \text{a) } \exists K \subset \Omega \text{ compact} : \operatorname{supp}(\phi_n) \subseteq K\ \forall n \in \mathbb{N} \text{ and} \\ & \text{b) } \partial^\alpha \phi_n \to 0 \text{ uniformly on } K\ \forall \text{ multi-indices } \alpha;\end{aligned}$$

$$\phi_n \to \phi \text{ in } C_0^\infty(\Omega) \iff \phi_n - \phi \to 0 \text{ in } C_0^\infty(\Omega).$$

The distributional space $\mathcal{D}'(\Omega)$ is then defined as the space of linear continuous functionals on $\mathcal{D}(\Omega)$. For such a functional $l \in \mathcal{D}'(\Omega)$, we adopt the notation

$$\langle l,\, \phi \rangle := l(\phi) \quad \forall \phi \in \mathcal{D}(\Omega),$$

and define its distributional derivative as

$$\left\langle \frac{\partial l}{\partial x_i},\ \phi \right\rangle := -\left\langle l, \frac{\partial \phi}{\partial x_i} \right\rangle \quad \forall \phi \in \mathcal{D}(\Omega).$$

We assume our differential operator L of degree m has the general shape

$$Lu = \sum_{i=0}^{m} \sum_{|\alpha|=i} l_\alpha \frac{\partial^{|\alpha|}}{\partial x^\alpha} u$$

with $\alpha = (\alpha_1, \alpha_2, \alpha_3)$ representing a multi-index of nonnegative integers and l_α the coefficient associated with the multi-index α. The defining relation (3.1) for the fundamental solution may then be rewritten

$$\sum_{i=0}^{m} (-1)^i \sum_{|\alpha|=i} l_\alpha \left\langle E(\cdot, y),\ \frac{\partial^{|\alpha|}}{\partial x^\alpha} \phi \right\rangle = \begin{cases} \phi(y), & y \in \Omega, \\ 0, & \text{else.} \end{cases}$$

After this brief excursion, let us give some examples of fundamental solutions of the Laplace operator in different dimensions.

Example 3.2.1. Let $L_x = -\triangle_x$. Then,

- in 1D, $E(x, y) = \frac{1}{2}(1 - |x - y|)$,
- in 2D, $E(x, y) = -\frac{1}{2\pi} \ln |x - y|$,
- in 3D, $E(x, y) = \frac{1}{4\pi} \frac{1}{|x-y|}$.

We have only given the definition of the fundamental solution for the three-dimensional case, but of course it is defined completely analogously in any dimension $d \in \mathbb{N}$. Nonetheless, as this example shows, the specific fundamental solutions in different dimensions may exhibit different forms. In particular, they vary in the degree of their singularity along the diagonal $E(x, x)$.

In order to demonstrate the relevance of the concept of fundamental solutions to the solution of partial differential equations, let us consider a simple application of fundamental solutions to the solution of differential equations.

Example 3.2.2. Consider the problem

$$L_x u = f \qquad \text{in } \mathbb{R}^3,$$

where L_x is a differential operator and u and f are functions in a suitable function space over $\mathbb{R}^3$. With a fundamental solution $E(x, y)$ of L_x, we may

define $u := E(\cdot, 0) * f$ (where $*$ represents the convolution operator) and obtain

$$L_x u = L_x(E(\cdot, 0) * f) = (L_x E(\cdot, 0)) * f = \\ = \delta(\cdot - 0) * f = \delta * f = f.$$

Therefore, u is a solution of the differential equation. Usually, though, we want to solve such equations in a bounded domain and to prescribe boundary conditions on the boundary of this domain; using this simple technique, there are no degrees of freedom left to fulfill such conditions. Clearly, we need more sophisticated methods for the solution of our problem, but this basic result might at least indicate the rationale behind the term "fundamental solution."

3.2.2 Green's identities

The third Green's identity will ultimately allow us to represent any value of the solution u within Ω in terms of the Cauchy data, $u|_\Gamma$ and $\frac{\partial u}{\partial n}|_\Gamma$.

We will first state the ubiquitous first Green's identity and use it as a starting point to obtain the remaining two identities. A more detailed derivation can be found in [11], as can the definition and the properties of the Hilbert spaces $H^k(\Omega)$ which we use without further introduction.

Theorem 3.2.1 (First Green's identity). *For all $u \in H^1(\Omega)$ and $v \in H^2(\Omega)$, we have*

$$\int_\Omega u \triangle v \, dx = \int_\Gamma u \frac{\partial v}{\partial n} \, ds - \int_\Omega \nabla u \cdot \nabla v \, dx. \tag{3.2}$$

By reversing the role of u and v in (3.2) and subtracting from the original formula, we lose the symmetric term and obtain the following result.

Theorem 3.2.2 (Second Green's identity). *For all $u, v \in H^2(\Omega)$, we have*

$$\int_\Omega (u \triangle v - v \triangle u) \, dx = \int_\Gamma \left(u \frac{\partial v}{\partial n} - v \frac{\partial u}{\partial n} \right) \, ds. \tag{3.3}$$

Finally, by choosing an arbitrary *source point* y from $\overline{\Omega}$ and inserting the fundamental solution of our differential operator into the second Green's identity via $v(x) = E(x, y)$, we can derive the third Green's identity or *representation formula.*

Theorem 3.2.3 (Third Green's identity). *For all $u \in H^2(\Omega)$ and all $y \in \overline{\Omega}$, we have*

$$\begin{aligned} \sigma(y) u(y) = &- \int_\Gamma u(x) \frac{\partial E(x, y)}{\partial n_x} \, ds_x + \int_\Gamma \frac{\partial u}{\partial n}(x) E(x, y) \, ds_x \\ &- \int_\Omega \triangle u(x) E(x, y) \, dx \end{aligned} \tag{3.4}$$

where

$$\sigma(y) = \lim_{\varepsilon \downarrow 0} \frac{1}{4\pi\varepsilon^2} \int_{x \in \Omega, |x-y|=\varepsilon} 1 \, ds_x.$$

Remark. The function σ in (3.4) has the following property:

$$\sigma(y) = \begin{cases} 1, & y \in \Omega, \\ \frac{1}{2}, & \text{almost everywhere on } \Gamma. \end{cases}$$

In particular, σ is equal to $\frac{1}{2}$ on every smooth part of the boundary.

It is important to note the justification for the term "representation formula." Given the values of $\triangle u$ within Ω as well as the full Cauchy data, $u|_\Gamma$ and $\frac{\partial u}{\partial n}|_\Gamma$, all terms on the right-hand side of (3.4) are known. Thus, we can then use this formula in order to calculate the term $\sigma(y)u(y)$ for all y; in particular, for $y \in \Omega$, we have $\sigma(y) \equiv 1$ and can thus evaluate the representation formula in order to obtain the value of the solution u at any desired point within Ω.

Of course, in typical boundary value problems, we are not given the full Cauchy data (as such a problem would be overdetermined), but only, in some sense, "half" of it. For instance, in the pure Dirichlet problem, we are given the full Dirichlet data, but none of the Neumann data; in some mixed boundary value problems, we are given the Dirichlet data on one part of the boundary and the Neumann data on the other one.

Our goal is therefore to obtain the unknown parts of the Cauchy data from the given parts. Once this is accomplished, the solution u may be evaluated at arbitrary points $y \in \Omega$. As mentioned before, however, in our model problem we are only interested in the Cauchy data itself.

Before we continue, we apply some simplifications in notation. As we will only be dealing with the Laplace equation, $\triangle u = 0$, we drop the last term in (3.4). Furthermore, we will henceforth use the abbreviation

$$v(y) := \frac{\partial u}{\partial n}(y) \quad \forall y \in \Gamma$$

for the normal derivative of the solution on the boundary. Hence we obtain the simplified representation formula

$$\sigma(y)u(y) = -\int_\Gamma u(x) \frac{\partial E(x,y)}{\partial n_x} \, ds_x + \int_\Gamma v(x) E(x,y) \, ds_x \quad \forall y \in \Omega'. \tag{3.5}$$

3.2.3 Boundary integral operators

In order to obtain a boundary integral equation involving only the Cauchy data from (3.5), we will choose the source point y from the boundary Γ. As $\sigma \equiv \frac{1}{2}$ almost everywhere on Γ, we will from now on also equate σ with $\frac{1}{2}$. This is not in general justified for approaches where the pointwise values of

σ must be taken into account. However, for the Galerkin method we will employ for discretization later on, only integrals over Γ will play a role, hence this is a reasonable simplification. We thus obtain

$$\frac{1}{2}u(y) = -\int_\Gamma u(x)\frac{\partial E(x,y)}{\partial n_x}\,ds_x + \int_\Gamma v(x)E(x,y)\,ds_x \quad \forall y \in \Gamma. \tag{3.6}$$

We now formally apply the differential operator $\frac{\partial}{\partial n_y}$ to this equation. It is not obvious that this is a mathematically sound operation, however it can be shown that the resulting equality does in fact hold. From this we get

$$\frac{1}{2}v(y) = -\int_\Gamma u(x)\frac{\partial^2 E(x,y)}{\partial n_x \partial n_y}\,ds_x + \int_\Gamma v(x)\frac{\partial E(x,y)}{\partial n_y}\,ds_x, \quad \forall y \in \Gamma. \tag{3.7}$$

The last two identities are integral equations over the boundary Γ involving four different integral operators. We introduce abbreviations for these four operators and then examine their mapping properties.

Definition 3.2.2. We define the boundary integral operators

$$\begin{aligned}
(Vv)(y) &:= \int_\Gamma E(x,y)v(x)\,ds_x, && \text{(single layer potential)}\\
(Ku)(y) &:= \int_\Gamma \frac{\partial E(x,y)}{\partial n_x}u(x)\,ds_x, && \text{(double layer potential)}\\
(K'v)(y) &:= \int_\Gamma \frac{\partial E(x,y)}{\partial n_y}v(x)\,ds_x, && \text{(adjoint double layer potential)}\\
(Du)(y) &:= -\int_\Gamma \frac{\partial^2 E(x,y)}{\partial n_x \partial n_y}u(x)\,ds_x. && \text{(hypersingular)}
\end{aligned}$$

The integrals must be interpreted in a weak sense; cf. [11].

Theorem 3.2.4. *The boundary integral operators from definition 3.2.2 are linear mappings between the following spaces:*

$$\begin{aligned}
V &: H^{-1/2}(\Gamma) \to H^{1/2}(\Gamma),\\
K &: H^{1/2}(\Gamma) \to H^{1/2}(\Gamma),\\
K' &: H^{-1/2}(\Gamma) \to H^{-1/2}(\Gamma),\\
D &: H^{1/2}(\Gamma) \to H^{-1/2}(\Gamma).
\end{aligned}$$

The single layer potential operator V is bounded and $H^{-1/2}(\Gamma)$-elliptic:

$$c_1^V \|w\|^2_{H^{-1/2}(\Gamma)} \le \langle Vw, w\rangle\,,\ \|Vw\|_{H^{1/2}(\Gamma)} \le c_2^V \|w\|_{H^{-1/2}(\Gamma)}\ \forall w \in H^{-1/2}(\Gamma).$$

The double layer potential operator K is bounded:

$$\|Kv\|_{H^{1/2}(\Gamma)} \le c_2^K \|v\|_{H^{1/2}(\Gamma)} \quad \forall v \in H^{1/2}(\Gamma).$$

The adjoint double layer potential operator K' is bounded:

$$\left\|K'w\right\|_{H^{-1/2}(\Gamma)} \leq c_2^{K'} \left\|w\right\|_{H^{-1/2}(\Gamma)} \quad \forall w \in H^{-1/2}(\Gamma).$$

The hypersingular operator D is bounded and $H^{1/2}(\Gamma)$-semielliptic:

$$c_1^D \left|v\right|^2_{H^{1/2}(\Gamma)} \leq \langle Dv, v\rangle\,,\ \left\|Dv\right\|_{H^{-1/2}(\Gamma)} \leq c_2^D \left\|v\right\|_{H^{1/2}(\Gamma)} \quad \forall v \in H^{1/2}(\Gamma).$$

Proof. The proofs for these statements are rather involved; see Appendix A.3 in [8].

Using these definitions, eqs. (3.6) and (3.7) may now be rewritten

$$\begin{aligned} \frac{1}{2}u &= Vv - Ku \qquad \text{on } \Gamma, \\ \frac{1}{2}v &= K'v + Du \qquad \text{on } \Gamma \end{aligned}$$

with $u \in H^{1/2}(\Gamma)$ and $v \in H^{-1/2}(\Gamma)$. By another simple reformulation, we get

$$\begin{pmatrix} u \\ v \end{pmatrix} = \underbrace{\begin{pmatrix} \frac{1}{2}I - K & V \\ D & \frac{1}{2}I + K' \end{pmatrix}}_{=:\mathcal{C}} \begin{pmatrix} u \\ v \end{pmatrix} \tag{3.8}$$

with the two by two block operator $\mathcal{C}$, the so-called *Calderón projector*. It can indeed be shown that $\mathcal{C}$ is a projector, that is, $\mathcal{C}^2 = \mathcal{C}$.

3.2.4 Boundary integral equations for the Laplace equation

We first state the standard mixed Dirichlet/Neumann boundary value problem for the Laplace equation.

Problem 3.2.1. Assume that we have a partitioning of the boundary into two nontrivial open subsets, $\Gamma = \overline{\Gamma}_D \cup \overline{\Gamma}_N$, $\Gamma_D \cap \Gamma_N = \emptyset$ with $|\Gamma_D| > 0$ and $|\Gamma_N| > 0$. Further assume that we are given the Dirichlet data $g_D \in H^{1/2}(\Gamma_D)$ and the Neumann data $g_N \in H^{-1/2}(\Gamma_N)$. We seek a function $u : \Omega \to \mathbb{R}$ such that

$$\begin{aligned} -\triangle u(x) &= f(x), \quad x \in \Omega, \\ u(x) &= g_D(x), \quad x \in \Gamma_D, \\ u(x) &= g_N(x), \quad x \in \Gamma_N. \end{aligned}$$

We define extensions of the Dirichlet and Neumann data to the entire boundary; in other words, we seek functions with the properties

$$\begin{aligned}\widetilde{g}_D &\in H^{1/2}(\Gamma), \quad \widetilde{g}_D|_{\Gamma_D} \equiv g_D,\\ \widetilde{g}_N &\in H^{-1/2}(\Gamma), \quad \widetilde{g}_N|_{\Gamma_N} \equiv g_N.\end{aligned}$$

For the full Dirichlet and Neumann data u and v, respectively, we choose the ansatz

$$\begin{aligned}u &= \widetilde{g}_D + u_N, \quad u_N \in H^{1/2}(\Gamma),\\ v &= \widetilde{g}_N + v_D, \quad v_D \in H^{-1/2}(\Gamma)\end{aligned}$$

with unknowns $u_N \in H^{1/2}(\Gamma)$ and $v_D \in H^{-1/2}(\Gamma)$. These ansatz functions for u and v are now inserted into (3.8). Exploiting the linearity of the boundary integral operators, we obtain the formulae

$$\begin{aligned}Vv_D - Ku_N - \frac{1}{2}u_N &= \frac{1}{2}\widetilde{g}_D + K\widetilde{g}_D - V\widetilde{g}_N \quad \text{on } \Gamma,\\ K'v_D + Du_N - \frac{1}{2}v_D &= \frac{1}{2}\widetilde{g}_N - D\widetilde{g}_D + K'\widetilde{g}_N \quad \text{on } \Gamma.\end{aligned}$$

This system of equations is overdetermined: intuitively speaking, we have one unknown per point $y \in \Gamma$, either $u_N(y)$ or $v_D(y)$ depending on the part of the boundary y is located on, but two full integral equations over Γ. There are multiple ways to proceed; for example, it is possible to completely drop the second equation and use only the first equation for solving the system. We will however choose an approach which leads to the so-called symmetric formulation of the boundary integral equation. This is done by restricting the first equation to Γ_D and the second to Γ_N. Recalling the definition of the extended boundary data above, we see that we have

$$\begin{aligned}u_N|_{\Gamma_D} &= u|_{\Gamma_D} - \widetilde{g}_D|_{\Gamma_D} = g_D - \widetilde{g}_D|_{\Gamma_D} = 0,\\ v_D|_{\Gamma_N} &= v|_{\Gamma_N} - \widetilde{g}_N|_{\Gamma_N} = v_N - \widetilde{g}_N|_{\Gamma_N} = 0\end{aligned} \tag{3.9}$$

and the restricted boundary integral equations thus take the form

$$\begin{aligned}Vv_D - Ku_N &= \frac{1}{2}\widetilde{g}_D + K\widetilde{g}_D - V\widetilde{g}_N \quad \text{on } \Gamma_D,\\ K'v_D + Du_N &= \frac{1}{2}\widetilde{g}_N - D\widetilde{g}_D + K'\widetilde{g}_N \quad \text{on } \Gamma_N.\end{aligned}$$

As all terms on the right-hand side are known, we introduce abbreviations for the sake of brevity,

$$\begin{aligned}f_D &:= \frac{1}{2}\widetilde{g}_D + K\widetilde{g}_D - V\widetilde{g}_N,\\ f_N &:= \frac{1}{2}\widetilde{g}_N - D\widetilde{g}_D + K'\widetilde{g}_N,\end{aligned}$$

and write

$$\begin{aligned} Vv_D - Ku_N &= f_D, \\ K'v_D + Du_N &= f_N. \end{aligned} \tag{3.10}$$

Our next aim is to solve this system of equations numerically by the use of an appropriate discretization scheme. There are several such schemes available. Widely used for its comparable ease of implementation is the *collocation method* where the integral equations are sought to be satisfied only in a discrete set of points located on the boundary, the so-called collocation points. While this method often yields satisfactory results in practice, there is no general theory of convergence available. We will therefore employ a Galerkin method which shares a common foundation with the schemes commonly used in finite element discretizations. This has the enormous advantage that the major convergence results from the very well-studied theory of Finite Element Methods become available for the treatment of the BEM as well.

3.3 Galerkin discretization

As in any Galerkin method, we first have to derive a variational formulation of the operator equation (3.10). We therefore introduce the trial space

$$\Lambda := \widetilde{H}^{-1/2}(\Gamma_D) \times \widetilde{H}^{1/2}(\Gamma_N)$$

from which candidate solutions (v_D, u_N) are to be drawn.

The spaces $\widetilde{H}^s$ used here are defined as follows. Let $\Gamma' \subset \Gamma$ be open. Then, for any $s \in \mathbb{R}_0^+$,

$$\begin{aligned} \widetilde{H}^s(\Gamma') &:= \left\{ v = \tilde{v}|_{\Gamma'} \,:\, \tilde{v} \in H^s(\Gamma),\, \operatorname{supp} \tilde{v} \subset \Gamma' \right\}, \\ \widetilde{H}^{-s}(\Gamma') &:= \left(H^s(\Gamma') \right)', \\ H^{-s}(\Gamma') &:= \left(\widetilde{H}^s(\Gamma') \right)'. \end{aligned}$$

In particular, this definition ensures that the $u_N \in \widetilde{H}^{1/2}(\Gamma_N)$ may be extended with zero to the entire boundary Γ, as required by (3.9).

We now take arbitrary test functions $(s,t) \in \Lambda$ and multiply the first and the second operator equation by s and t respectively.

$$\begin{aligned} \langle Vv_D, s \rangle_{\Gamma_D} - \langle Ku_N, s \rangle_{\Gamma_D} &= \langle f_D, s \rangle_{\Gamma_D}, \\ \langle K'v_D, t \rangle_{\Gamma_N} + \langle Du_N, t \rangle_{\Gamma_N} &= \langle f_N, t \rangle_{\Gamma_N}. \end{aligned}$$

The products used here are the duality products

$$\begin{aligned} \langle \cdot, \cdot \rangle_{\Gamma_D} &: H^{1/2}(\Gamma_D) \times \widetilde{H}^{-1/2}(\Gamma_D) \to \mathbb{R}, \\ \langle \cdot, \cdot \rangle_{\Gamma_N} &: \widetilde{H}^{-1/2}(\Gamma_N) \times H^{1/2}(\Gamma_N) \to \mathbb{R} \end{aligned}$$

since $\widetilde{H}^{1/2}(\Gamma_D) \subset H^{1/2}(\Gamma_D)$ and $\widetilde{H}^{1/2}(\Gamma_N) \subset H^{1/2}(\Gamma_N)$. We again refer to [11] for a more in-depth discussion of the involved spaces.

We can thus represent the operator equation (3.10) in a variational form.

Problem 3.3.1. Find $(v_D, u_N) \in \Lambda$ such that for all $(s,t) \in \Lambda$, there holds

$$a(u_N, v_D; s, t) = F(s,t)$$

with

$$\begin{aligned} a(u,v;s,t) =& \langle Vv, s\rangle_{\Gamma_D} - \langle Ku, s\rangle_{\Gamma_D} + \left\langle K'v, t\right\rangle_{\Gamma_N} + \langle Du, t\rangle_{\Gamma_N}, \\ F(s,t) =& \frac{1}{2}\langle \widetilde{g}_D, s\rangle_{\Gamma_D} + \langle K\widetilde{g}_D, s\rangle_{\Gamma_D} - \langle V\widetilde{g}_N, s\rangle_{\Gamma_D} \\ &+ \frac{1}{2}\langle \widetilde{g}_N, t\rangle_{\Gamma_N} - \langle D\widetilde{g}_D, t\rangle_{\Gamma_N} - \left\langle K'\widetilde{g}_N, t\right\rangle_{\Gamma_N}. \end{aligned}$$

Note that we have combined both lines of the equation into one here. Of course, by setting either of the test functions s and t to zero, we can reconstruct the individual lines of the original equation.

The main concept behind the Galerkin method is now to choose a finite-dimensional subspace $\Lambda_h \subset \Lambda$ of our trial space. To this end, we introduce a triangular mesh approximating the boundary Γ consisting of $N_T \in \mathbb{N}$ triangular boundary elements $(\tau_k)_{k=1,\dots,N_T}$.

$$\Gamma \approx \Gamma_h = \bigcup_{k=1}^{N_T} \overline{\tau}_k$$

The vertices of the mesh are denoted by $(x_i)_{i=1,\dots,N_V}$. Without loss of generality, we assume that the triangles are ordered in such a way that the first $N_{DT} < N_T$ boundary elements make up the Dirichlet part of the boundary, that is,

$$\Gamma_h = \Gamma_{Dh} \cup \Gamma_{Nh} \quad \text{where}$$
$$\Gamma_{Dh} = \bigcup_{k=1}^{N_{DT}} \overline{\tau}_k, \quad \Gamma_{Nh} = \bigcup_{k=N_{DT}+1}^{N_T} \overline{\tau}_k.$$

Similarly, we assume that the first $N_{NV} < N_V$ vertices are those which are located on the interior of the Neumann boundary:

$$x_i \in \begin{cases} \mathring{\Gamma}_{Nh}, & 1 \le i \le N_{NV}, \\ \overline{\Gamma}_{Dh}, & N_{NV} < i \le N_V. \end{cases}$$

Let $|\tau_k|$ denote the area of the triangle τ_k. We then define the local mesh size

$$h_k := \sqrt{|\tau_k|},$$

the global mesh sizes

$$h = h_{\max} = \max_{1 \le k \le N_T} h_k, \quad h_{\min} = \min_{1 \le k \le N_T} h_k$$

and the element diameter

$$d_k := \sup_{x,y \in \tau_k} |x - y| .$$

Assume that the mesh Γ_h is drawn from a family of meshes $(\Gamma_h)_{h>0}$ with corresponding mesh sizes h. We then require this family to be

- *globally quasi-uniform*, that is, the ratio

  $$\frac{h_{\max}}{h_{\min}} \le c_G$$

  should be bounded by a number $c_G > 0$ independent of the mesh size, and

- *uniformly shape-regular*, that is, we want to have

  $$d_k \le c_B h_k \qquad \forall k \in \{1, \dots, N_T\}$$

  with some number $c_B > 0$ which is independent of the mesh size.

For the Neumann data, we now define basis functions

$$\psi_k := \chi_{\tau_k} \qquad \forall k \in \{1, \dots, N_T\}$$

where χ_{τ_k} denotes the characteristic function of the triangle τ_k. In other words, ψ_k is the discontinuous, piecewise constant function which is identically 1 on τ_k and identically 0 everywhere else.

For the Dirichlet data, we define basis functions ϕ_i characterized by

$$\phi_i|_{\tau_k} \in P_1(\tau_k), \quad \phi_i(x_j) = \delta_{ij} \quad \forall i, j \in \{1, \dots, N_V\}, k \in \{1, \dots, N_T\}$$

where δ_{ij} denotes the Kronecker delta. In other words, ϕ_i is the continuous, piecewise linear function which is 1 in x_i and 0 in all other points x_j, $j \neq i$.

Our finite-dimensional trial spaces are thus given by

$$\begin{aligned} S_h^0(\Gamma_D) &:= \operatorname{span} \{\psi_k\}_{k=1}^{N_{DT}} \subset \widetilde{H}^{-1/2}(\Gamma_D), \\ S_h^1(\Gamma_N) &:= \operatorname{span} \{\phi_i\}_{i=1}^{N_{NV}} \subset \widetilde{H}^{1/2}(\Gamma_N), \\ \Lambda_h &:= S_h^0(\Gamma_D) \times S_h^1(\Gamma_N) \subset \Lambda \end{aligned}$$

with the dimensions

$$\dim \Lambda_h = \dim S_h^0(\Gamma_D) + \dim S_h^1(\Gamma_N) = N_{DT} + N_{NV} = \mathcal{O}\left(h^{-2}\right).$$

We have the approximation properties (see [8])

$$\forall w \in H^s_{\mathrm{pw}}(\Gamma),\ s \in [0,1],\ \sigma \in [-1,0]:$$
$$\inf_{w_h \in S^0_h(\Gamma)} \|w - w_h\|_{H^\sigma(\Gamma)} \leq c h^{s-\sigma} |w|_{H^s_{\mathrm{pw}}(\Gamma)} \quad (3.11)$$

for the piecewiese constant basis functions and

$$\forall v \in H^s_{\mathrm{pw}}(\Gamma),\ s \in [1,2],\ \sigma \in [-2,1]:$$
$$\inf_{v_h \in S^1_h(\Gamma)} \|v - v_h\|_{H^\sigma(\Gamma)} \leq c h^{s-\sigma} |v|_{H^s_{\mathrm{pw}}(\Gamma)} \quad (3.12)$$

for the piecewise linear basis functions. Here $H^s_{\mathrm{pw}}(\Gamma)$ is the space of all piecewise (per triangle) H^s functions on Γ for $s \geq 0$, that is,

$$H^s_{\mathrm{pw}}(\Gamma) := \{v \in L_2(\Gamma) : v|_{\tau_k} \in H^s(\tau_k),\ k = 1, \ldots, N_T\}$$

and $\|v\|^2_{H^s_{\mathrm{pw}}(\Gamma)} := \sum_{k=1}^{N_T} \|v|_{\tau_k}\|^2_{H^s(\tau_k)}$.

It is now trivial to formulate a discretized version of problem 3.3.1.

Problem 3.3.2. Find $(v_{Dh}, u_{Nh}) \in \Lambda_h$ such that for all $(s_h, t_h) \in \Lambda_h$, there holds

$$a(u_{N_h}, v_{Dh}; s_h, t_h) = F(s_h, t_h).$$

By Céa's lemma and application of the Aubin-Nitsche trick as well as an inverse equality argument [8], the following error estimates are obtained.

Theorem 3.3.1 (Galerkin BEM error estimate). *If $u \in H^s(\Omega)$ for some $s \in [1, \frac{5}{2}]$ is the solution of the boundary value problem 3.2.1, (v_D, u_N) is the exact solution of the variational problem 3.3.1 and (v_{Dh}, u_{Nh}) is the solution of problem 3.3.2, then, for any $\sigma \in [-2, 0]$, we have*

$$\|v_D - v_{Dh}\|^2_{H^\sigma(\Gamma_D)} + \|u_N - u_{Nh}\|^2_{H^{\sigma+1}(\Gamma_N)} \leq c h^{2(s-\sigma)-3} |u|^2_{H^s(\Omega)}.$$

In particular, for $u \in H^2(\Omega)$, we obtain

$$\|v_D - v_{Dh}\|_{L_2(\Gamma_D)} \leq c h^{1/2} |u|_{H^2(\Omega)}, \quad (3.13)$$
$$\|u_N - u_{Nh}\|_{L_2(\Gamma_N)} \leq c h^{3/2} |u|_{H^2(\Omega)}. \quad (3.14)$$

3.4 Calculation of the system matrices

The next step towards solving a boundary value problem using a BEM is rewriting problem 3.3.2 in the form of a matrix equation and calculating the entries of the system matrix. This system can then be tackled using a suitable method for the solution of linear systems.

First we write down an explicit representation of the discretized solution, namely

$$u_{Nh} = \sum_{j=1}^{N_{NV}} u_j \phi_j, \qquad v_{Dh} = \sum_{\ell=1}^{N_{DT}} v_\ell \psi_\ell.$$

We insert these representations into problem 3.3.2. Because of the linearity of all involved operators, it is sufficient to test with the basis functions $(\psi_k)_{k=1}^{N_{DT}}$ and $(\phi_i)_{i=1}^{N_{NV}}$ instead of general test functions $(s_h, t_t) \in \Lambda_h$. This yields

$$\sum_{\ell=1}^{N_{DT}} \langle V\psi_\ell, \psi_k \rangle_{\Gamma_D} v_\ell - \sum_{j=1}^{N_{NV}} \langle K\phi_j, \psi_k \rangle_{\Gamma_D} u_j = F(\psi_k, 0), \quad k = 1, \ldots, N_{DT},$$

$$\sum_{\ell=1}^{N_{DT}} \langle K'\psi_\ell, \phi_i \rangle_{\Gamma_N} v_\ell + \sum_{j=1}^{N_{NV}} \langle D\phi_j, \phi_i \rangle_{\Gamma_N} u_j = F(0, \phi_i), \quad i = 1, \ldots, N_{NV}$$

(where we have separated the two equations again). This may be written in matrix notation as

$$\begin{pmatrix} V_h & -K_h \\ K_h^T & D_h \end{pmatrix} \begin{pmatrix} \underline{v}_{Dh} \\ \underline{u}_{Nh} \end{pmatrix} = \begin{pmatrix} \underline{f}_{Dh} \\ \underline{f}_{Nh} \end{pmatrix} \tag{3.15}$$

with the entities

$$\begin{aligned} V_h &\in \mathbb{R}^{N_{DT} \times N_{DT}}, \\ K_h &\in \mathbb{R}^{N_{DT} \times N_{NV}}, \\ D_h &\in \mathbb{R}^{N_{NV} \times N_{NV}}, \\ \underline{v}_{Dh}, \underline{f}_{Dh} &\in \mathbb{R}^{N_{DT}}, \\ \underline{u}_{Nh}, \underline{f}_{Nh} &\in \mathbb{R}^{N_{NV}} \end{aligned}$$

given by

$$\begin{aligned} \underline{v}_{Dh} &= (v_k)_{k=1}^{N_{DT}}, \\ \underline{u}_{Nh} &= (u_i)_{i=1}^{N_{NV}}, \\ V_h[k,\ell] &= \frac{1}{4\pi} \int_{\tau_k} \int_{\tau_\ell} \frac{1}{|x-y|} \, ds_y \, ds_x, && (3.16) \\ K_h[k,j] &= \frac{1}{4\pi} \int_{\tau_k} \int_{\Gamma_N} \frac{(x-y, n(y))}{|x-y|^3} \phi_j(y) \, ds_y \, ds_x, && (3.17) \\ D_h[i,j] &= \frac{1}{4\pi} \int_{\Gamma_N} \int_{\Gamma_N} \frac{(\underline{\mathrm{curl}}_\Gamma \phi_j(y), \underline{\mathrm{curl}}_\Gamma \phi_i(y))}{|x-y|} \, ds_y \, ds_x. && (3.18) \end{aligned}$$

This representation for the entries of D_h is obtained by applying integration by parts to the original definition of D, described in detail in [8]. We use here the *surface curl* operator defined as

$$\underline{\mathrm{curl}}_\Gamma u(x) = n(x) \times \nabla_x \tilde{u}(x) \quad \text{for } x \in \Gamma$$

where $\tilde{u}$ denotes a locally defined extension of u into the neighborhood of Γ. Noting that $\underline{\mathrm{curl}}_\Gamma \phi_i$ is piecewise constant per triangle, we see that the entries of D_h may in fact be computed as a straightforward linear combination of the entries of V_h.

In order to compute the discretized right-hand side, we first approximate the given Cauchy data within the finite-dimensional space Γ_h, that is,

$$\tilde{g}_D \approx \tilde{g}_{Dh} = \sum_{i=N_{NV}+1}^{N_V} g_{Di}\phi_i,$$
$$\tilde{g}_N \approx \tilde{g}_{Nh} = \sum_{k=N_{DT}+1}^{N_T} g_{Nk}\psi_k.$$

We then insert these approximations into the equation for the right-hand side and obtain

$$\begin{pmatrix} \underline{f}_{Dh} \\ \underline{f}_{Nh} \end{pmatrix} = \begin{pmatrix} \frac{1}{2}\bar{M}_{1h} + \bar{K}_{1h} & \bar{V}_h \\ \bar{D}_h & \frac{1}{2}\bar{M}_{2h}^T + \bar{K}_{2h}^T \end{pmatrix} \begin{pmatrix} \underline{g}_{Dh} \\ \underline{g}_{Nh} \end{pmatrix} \tag{3.19}$$

with

$$\begin{aligned}
\underline{g}_{Dh} &= (g_{Di})_{i=N_{NV}+1,\ldots,N_V}, \\
\underline{g}_{Nh} &= (g_{Nk})_{k=N_{DT}+1,\ldots,N_T}, \\
\bar{M}_{1h}[k,j] &= \int_{\tau_k} \phi_j(x)\, ds_x, \quad 1 \le k \le N_{DT},\ N_{NV} < j \le N_V, \\
\bar{V}_h[k,\ell] &= \langle V\psi_\ell, \psi_k \rangle, \quad 1 \le k \le N_{DT},\ N_{DT} < \ell \le N_T, \\
\bar{K}_{1h}[k,j] &= \langle K\phi_i, \psi_k \rangle, \quad 1 \le k \le N_{DT},\ N_{NV} < j \le N_V, \\
\bar{M}_{2h}[k,j] &= \int_{\tau_k} \phi_j(x)\, ds_x, \quad N_{DT} < k \le N_T,\ 1 \le j \le N_{NV}, \\
\bar{K}_{2h}[k,j] &= \langle K\phi_i, \psi_k \rangle, \quad N_{DT} < k \le N_T,\ 1 \le j \le N_{NV}, \\
\bar{D}_h[i,j] &= \langle D\phi_j, \phi_i \rangle, \quad 1 \le i \le N_{NV},\ N_{NV} < j \le N_V.
\end{aligned}$$

The entries may be evaluated in the same form as in eqs. (3.16)–(3.18). The difference to those equations lies merely in the ranges of indices on which the matrices are defined.

The entries of the BEM matrices cannot simply be computed by the use of a quadrature rule because of the singular kernels of the integral operators. A combination of numerical and analytic methods is commonly used for the evaluation of these integrals. We refer the reader to [8] for a detailed derivation of such schemes. Also, there are software libraries available which can compute these entries in a black-box fashion (see [10]), and such a library was used for the software implementation described in Chapter 5.

3.5 Properties of the Galerkin BEM matrix

From inspection (and as a result of the symmetry of the underlying operators), it is clear that both matrices V_h and D_h are symmetric. Also, the ellipticity of the single layer potential operator V translates directly into positive definiteness of the matrix V_h.

The question remains which method to choose in order to solve this system. In any such decision, the properties of the system matrix of course play a principal role. In our case, an inspection of the boundary element integrals reveals that we cannot hope for a sparse matrix structure as is the case in a comparable Finite Element Method. In general, we will therefore have to contend with a linear system with $\dim \Lambda_h = \mathcal{O}\left(h^{-2}\right)$ unknowns and a full matrix with $\mathcal{O}\left(h^{-4}\right)$ non-zero entries. Solution of such a system via a standard direct solver takes on the order of $\mathcal{O}\left(h^{-6}\right)$ operations. Clearly, this is intractable even for moderate-sized problems. Commonly used iterative methods have similar problems: even if we were to find a method which converges in $\mathcal{O}(1)$ iterations, we still have to perform at least one matrix-vector product per iteration, which makes the total number of operations at least $\mathcal{O}\left(h^{-4}\right)$. This is still quadratic in the number of unknowns and hence not suitable for large-scale problems.

If we compare this to the use of a Finite Element Method, we see that in this case we have to discretize the entire computational domain Ω, thus resulting in $\mathcal{O}\left(h^{-3}\right)$ unknowns for a mesh size of h. Although this means that a FEM has to operate on a number of unknowns which is larger by an order of h^{-1}, there are quasi-optimal solvers available which solve such FEM systems in slightly more than $\mathcal{O}\left(h^{-3}\right)$ operations, thus being more efficient than the basic BEM approach outlined above.

This situation has long posed a hindrance for the application of the BEM to three-dimensional problems of any relevant size. Relatively recent developments have however produced techniques which do allow the solution of such systems in a reasonable timeframe. The core idea behind these methods is the approximation of the system matrix by so-called data-sparse matrices, in particular by hierarchical matrix representations. The ultimate goal is the solution of BEM systems in a quasi-optimal manner, that is, with only slightly more than $\mathcal{O}\left(h^{-2}\right)$ operations. We investigate these techniques in Chapter 4.

Chapter 4

Data-sparse Matrix Approximation

The basic concept behind data-sparse matrix approximation is to approximate a large matrix by another matrix which requires significantly fewer data points to represent. Similarly, typical matrix operations (e.g. applying the matrix to a given vector or solving a linear system with the matrix as its coefficient matrix) should require significantly less numerical effort for the approximated matrix. In particular, the techniques presented in this chapter are built upon two ideas: hierarchical representation and low-rank approximation of matrices.

As this chapter deals a lot with matrices of a given maximum rank, we introduce notation for such classes of matrices.

Definition 4.0.1. Let $m, n \in \mathbb{N}$, $k \in \mathbb{N}_0$. We denote the set of all m-by-n matrices with rank at most k by

$$\mathcal{R}(k, m, n) := \{M \in \mathbb{R}^{m \times n} : \text{rank } M \leq k\}.$$

Note that for a matrix $A \in \mathcal{R}(k, m, n)$, there is always a representation as a sum of rank one outer products of vectors, i. e.

$$A = \sum_{i=1}^{k} u_i v_i^T$$

with vectors $u_i \in \mathbb{R}^m,\ v_i \in \mathbb{R}^n$. Such a representation requires $km + kn = k(m + n)$ floating point storage cells, as opposed to storing all mn entries of the full matrix A. Thus, assuming for simplicity that $m = n$, if we can find a low-rank approximation of reasonable accuracy with rank $k \ll \frac{n}{2}$, we save a lot of space and can also perform the matrix-vector product Ax in $\mathcal{O}(k(m + n))$ operations.

Not all matrices permit such an efficient approximation. In section 4.1, we will investigate experimentally some cases to determine when this may

be possible. Section 4.2 introduces the concept of hierarchical matrices. In section 4.3, we study various methods to approximate matrix blocks in a low-rank fashion. Section 4.4 introduces some operations which may be performed on hierarchical matrices. Finally, in section 4.5, we investigate how these methods may be applied to the BEM system derived in Chapter 3.

The exposition in this chapter follows quite closely that in the excellent lecture notes by Börm, Grasedyck and Hackbusch [4]. For a more detailed discussion of the topics presented here, we also refer the reader to the monograph [2] by Bebendorf.

4.1 An introductory example

For our first experiments in approximating matrices by a low-rank representation, we use the truncated singular value decomposition. It is well known that any m-by-n matrix A can be represented in the form

$$A = U\Sigma V^T$$

with orthogonal matrices $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$ as well as the matrix $\Sigma \in \mathbb{R}^{m\times n}$ with the entries

$$\Sigma_{ij} = \begin{cases} \sigma_i, & \text{for } i = j < \text{rank } A, \\ 0, & \text{else.} \end{cases}$$

The real numbers $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_{\text{rank } A} > 0$ are called the singular values of A. The vectors $u_i \in \mathbb{R}^m$, $1 \leq i \leq m$ and $v_j \in \mathbb{R}^n$, $1 \leq j \leq n$ which make up the columns of U and V respectively are called singular vectors of A.

Definition 4.1.1. Given a singular value decomposition $U\Sigma V^T$ of the matrix $A \in \mathbb{R}^{m\times n}$, the *truncated singular value decomposition* with rank $r \in \{1, \ldots, \text{rank } A\}$ of A is given by

$$\tilde{A}(r) := U\tilde{\Sigma}(r)V^T \in \mathcal{R}(r, m, n)$$

with

$$\tilde{\Sigma}(r)_{ij} = \begin{cases} \Sigma_{ij}, & i = j \leq r, \\ 0, & \text{else.} \end{cases}$$

In other words, the truncated singular value decomposition (TSVD for short) disregards all but the largest r singular values. Note that we can equivalently write

$$\tilde{A}(r) = \sum_{i=1}^{r} \sigma_i u_i v_i^T$$

and thus the TSVD fits neatly into the framework for low-rank matrix approximation outlined in the introduction of this chapter.

It can be shown that $\tilde{A}(r)$ is the best approximation of rank r to A in the sense that it minimizes the error in the Frobenius norm. That is,

$$\left\|A - \tilde{A}(r)\right\|_F = \min\left\{\|A - B\|_F : B \in \mathcal{R}(r, m, n)\right\},$$
$$\|B\|_F = \left(\sum_{i=1}^{m}\sum_{j=1}^{n} B_{ij}^2\right)^{\frac{1}{2}} \qquad \text{for any } B \in \mathbb{R}^{m\times n}.$$

In fact, by using the fact that the Frobenius norm is unitarily invariant, it is easy to calculate the exact approximation error:

$$\left\|A - \tilde{A}(r)\right\|_F^2 = \left\|U(\Sigma - \tilde{\Sigma}(r))V^T\right\|_F^2 = \left\|\Sigma - \tilde{\Sigma}(r)\right\|_F^2 = \sum_{i=r+1}^{\operatorname{rank} A} \sigma_i^2.$$

Therefore, analyzing the magnitude of the singular values of some matrix gives us a very good idea about how well it may be approximated by a low-rank matrix via the use of the TSVD. In the following, we perform such an analysis on a matrix which shares some crucial features with the BEM system matrices given in Chapter 3.

We assume that we are given a kernel function

$$K : [0,1]^2 \to \mathbb{R},$$
$$(x, y) \mapsto \frac{1}{\alpha + |x - y|^2}$$

with some small parameter $\alpha > 0$, a regular grid of $m \cdot n$ $(m, n \in \mathbb{N})$ discretization points

$$\left\{(x_i, y_j) \in [0,1]^2 : x_i = \frac{i-1}{m-1}, y_j = \frac{j-1}{n-1},\ i \in \{1,\dots,m\}, j \in \{1,\dots,n\}\right\}$$

and a matrix $A \in \mathbb{R}^{m\times n}$ obtained by discretizing the kernel K,

$$A_{ij} = K(x_i, y_j), \qquad i \in \{1,\dots,m\}, j \in \{1,\dots,n\}.$$

Note that the matrix A obtained in this way is similar in structure to the BEM matrices given in eqs. (3.16)–(3.18), although for simplicity we have employed here a simple Nyström-like discretization scheme instead of the Galerkin discretization used in Chapter 3. The parameter $\alpha > 0$ is thus required to obtain finite pointwise values on the diagonal of the kernel K; for our examples, we choose $\alpha = 10^{-4}$. Figure 4.1 gives a sample plot of the entries of the matrix A for $m = n = 32$; each colored box corresponds to one entry of the matrix, and the color of each box was chosen from a colormap according to the logarithmic value of the corresponding matrix entry. The

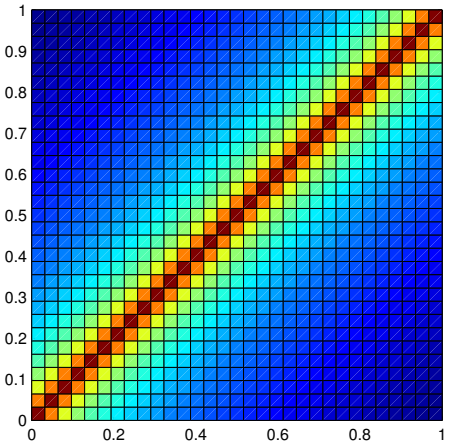

Figure 4.1: A logarithmically color-coded plot of the matrix A for $m = n = 32$, $\alpha = 10^{-4}$.

near-singularity along the diagonal can be plainly seen, as can the rapidly decaying entries as the distance to the diagonal increases.

In order to determine the quality of a low-rank TSVD approximation of this matrix, we calculate the singular values and display them in a semi-logarithmic plot, given in fig. 4.2 for $m = n = 1024$.

As we can see, both the singular values and the relative error for the TSVD approximation decrease exponentially. However, we recall that we require a truncation index of $r \ll \frac{n}{2}$ for an efficient low-rank representation; let us for the sake of argument fix $r = \frac{n}{4} = 256$. The right-hand plot in fig. 4.2 then indicates that we have to contend with a relative error on the order of $\left\|A - \tilde{A}(r)\right\|_F \approx 10^{-4}$. This is clearly inacceptable for the solution of a linear system with A as its coefficient matrix when we take the condition number $\kappa(A) \approx 10^{13}$ into account.

It is conceivable that the strong singularity along the diagonal of A is to blame for the poor results. The starting point for the technique of hierarchical matrix approximation is thus the following idea: we split the matrix A into several submatrices such that at least some of them can be approximated efficiently. Let us explore this idea experimentally.

We split the matrix A into four equally-sized blocks. Instead of the full matrix A, we now examine only one of the two off-diagonal blocks. The resulting matrix is visualized in fig. 4.3.

The singularity now only occurs in one corner of this submatrix instead of along the diagonal. We calculate the singular values and the TSVD approximation error for this matrix block and display them in fig. 4.4.

Indeed, the splitting of the matrix results in much more favourable be-

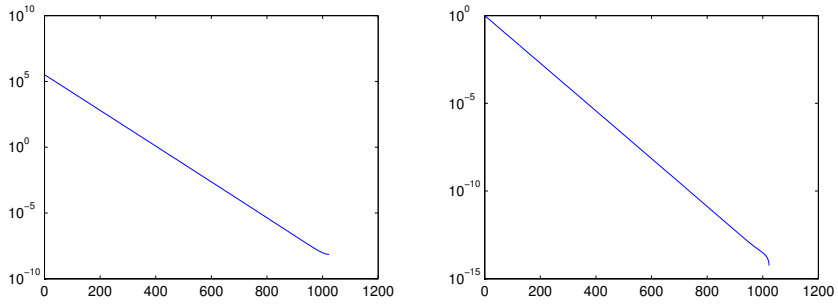

Figure 4.2: Left: Semi-logarithmic plot of the singular values of A.
Right: Semi-logarithmic plot of the relative error for TSVD with varying truncation index r.
Both plots were created with $m = n = 1024$.

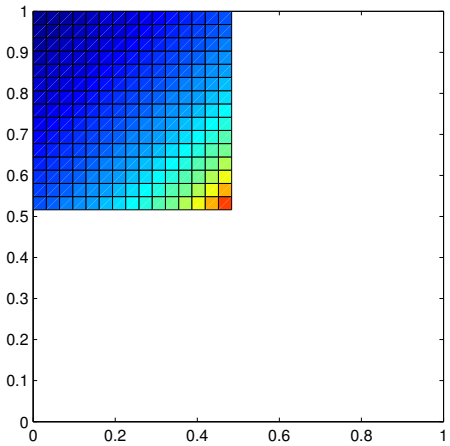

Figure 4.3: A logarithmically color-coded plot of one off-diagonal block of A for $m = n = 32$, $\alpha = 10^{-4}$.

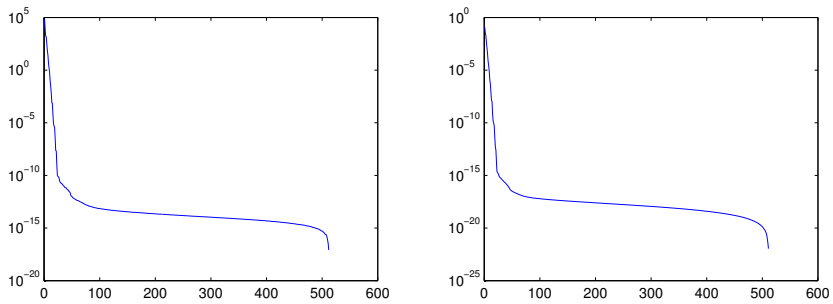

Figure 4.4: Left: Semi-logarithmic plot of the singular values of one off-diagonal block of A.
Right: Semi-logarithmic plot of the relative error for TSVD with varying truncation index r.
Both plots were created with $m = n = 1024$, and thus a submatrix of size 512.

haviour of the singular values; we can see that only very few singular values are significant for the approximation of the submatrix. Fixing again r at a quarter of the matrix size, i. e. in this case $r = 128$, we obtain for the TSVD approximation a relative error on the order of 10^{-18}, a much more reasonable error than for the full matrix.

Hence, we are now able to approximate the two off-diagonal blocks of A in an efficient manner by a low-rank representation. The two diagonal blocks of A, however, resist such treatment as they possess qualitatively the same structure as A, having a singularity along the diagonal. However, we can apply the idea of subdivision recursively: we again split each of the diagonal blocks into four smaller blocks, thus obtaining new diagonal and off-diagonal blocks. The latter may again be approximated efficiently as outlined above for the first level of subdivision. If necessary, the new, smaller diagonal blocks are subdivided recursively in this fashion until the parts of the matrix which can not be approximated well are small enough to be stored in a full matrix format. This is the key concept of hierarchical matrix approximation.

Let us carry out such a scheme for the matrix A with $m = n = 256$ for the purpose of demonstration. We also choose now $\alpha = 2^{-9}$ and a desired approximation error $\varepsilon = 10^{-6}$ for each matrix block. The result of three iterations of recursive subdivision can be seen in fig. 4.5. Here, the numbers in the matrix blocks indicate the rank r of a TSVD which would be required to approximate the corresponding block to an accuracy of ε. The blocks whose approximation rank exceeds a given maximum rank $r_{\max} = 12$ are shown in red; these are the blocks which will be subdivided again in the next iteration. The blocks who may be approximated accurately enough with $r < r_{\max}$ are shown in green and are not modified further in the next

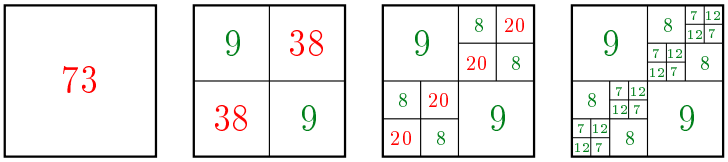

Figure 4.5: A recursive approximation scheme for $m = n = 256$.

iteration.

After three iterations, this scheme has produced a hierarchical matrix where all blocks require a rank less or equal to $r_{\max}$ to approximate. The rightmost matrix in fig. 4.5 requires 18,432 words of memory to store, as opposed to the $256^2 = 65,536$ words of memory required for the original full matrix A. This corresponds roughly to a reduction to a quarter of the original memory usage.

Clearly, this simple subdivision scheme is tailored quite specifically to the simple structure of the matrix used as an example here. It is however possible to generalize the concept to matrices obtained by the discretization of higher-dimensional kernels. The next section is concerned with such a generalization.

4.2 Hierarchical matrix subdivision

4.2.1 Cluster trees

As the structure of a hierarchical matrix subdivision is essentially that of a tree, we introduce some notation for dealing with trees. We will use most of the properties of trees in a rather intuitive fashion; see [4] for a more rigorous exposition with proofs for all nontrivial tree properties.

Definition 4.2.1 (Trees). Let $N \neq \emptyset$ be a finite set, let $r \in N$ and let $S : N \to \mathcal{P}(N)$ be a mapping from N into subsets of N. For any $t \in N$, a sequence $(t_0, \ldots, t_m)$ with $t_0 = r$, $t_m = t$ and $t_{i+1} \in S(t_i)$ for $0 \leq i < m$ is called a *sequence of ancestors* of t.

The triple $T = (N, r, S)$ is called a *tree* if and only if there is exactly one sequence of ancestors for every $t \in N$.

For a tree $T = (N, r, S)$, the elements of N are called *nodes*, the element r is called the *root node* or simply *root* and is denoted by $r = \operatorname{root}(T)$, and the set $\operatorname{sons}_T(t) := S(t)$ is called the set of *sons* of t. The subscript T may be omitted if the tree in question is clear from the context.

We define the *set of descendants* of a node $t \in N$ recursively as follows:

$$\operatorname{sons}^*(t) := \begin{cases} \{t\}, & \text{if } \operatorname{sons}(t) = \emptyset, \\ \{t\} \cup \bigcup_{s \in \operatorname{sons}(t)} \operatorname{sons}^*(s), & \text{otherwise.} \end{cases}$$

A node $t \in N$ is called a *leaf* of T if and only if $\text{sons}(t) = \emptyset$. The set of leaves is thus defined as

$$\mathcal{L}(T) := \{t \in N : \text{sons}(t) = \emptyset\}.$$

Definition 4.2.2 (Tree level). Let $T = (N, r, s)$ be a tree. Let $t \in N$, and let $(t_0, \ldots, t_m)$ be its (by definition uniquely determined) sequence of ancestors. Then the number $m \in \mathbb{N}_0$ is called the *level* of the node t, and we denote

$$\text{level}_T(t) := m$$

where again the subscript T may be omitted where appropriate. The *depth* of the tree T is defined as

$$\text{depth}(T) := \max\{\text{level}(t) : t \in N\}.$$

Definition 4.2.3 (Labeled trees). Let (N, r, S) be a tree. Furthermore, let $L \neq \emptyset$ be a finite set and let $m : N \to L$ be a mapping. Then,

$$T = (N, r, S, m, L)$$

is called a *labeled tree* and L is called its *label set*. For each node $t \in N$, $m(t)$ is called the *label* of t and is abbreviated by $\hat{t} := m(t)$.

Let us assume that we have a family of basis functions

$$\phi_i : \mathbb{R}^d \to \mathbb{R}, \qquad i \in \mathcal{I}$$

with an index set $\mathcal{I}$. Typically, the index set would simply consist of a contiguous sequence of integers, e.g. $\mathcal{I} = \{1, \ldots, m\}$, although we do not require this here. Also, the basis functions $(\phi_i)_{i \in \mathcal{I}}$ are not specified in more detail here, but in practice the piecewise constant or continuous piecewise linear basis functions introduced earlier are common choices. Other choices are possible, but we do require the basis functions to possess local support $\text{supp}(\phi_i)$.

We now define a structure which represents a hierarchical subdivision of such a family of basis functions represented by their index set $\mathcal{I}$.

Definition 4.2.4 (Cluster trees). A labeled tree $T_{\mathcal{I}} = (N, r, S, m, \mathcal{P}(\mathcal{I}))$ is called a *cluster tree* for $\mathcal{I}$ if and only if the following conditions are satisfied:

- $\widehat{\text{root}(T_{\mathcal{I}})} = \mathcal{I}$,
- for all $t \in N$ with $\text{sons}(t) \neq \emptyset$, we have

$$\hat{t} = \bigcup_{s \in \text{sons}(t)} \hat{s},$$

- for all $t \in N$ and all $s_1, s_2 \in \text{sons}(t)$ with $s_1 \neq s_2$, we have

$$\hat{s}_1 \cap \hat{s}_2 = \emptyset.$$

The nodes $t \in N$ of a cluster tree are called *clusters*, and for short we simply write $t \in T_{\mathcal{I}}$.

Remark. The previous definition may be summarized in words as follows: every node t of a cluster tree is associated with some subset of the complete index set, $\hat{t} \subset \mathcal{I}$. The root of a cluster tree represents the complete index set, and the sons of every non-leaf node t form a partition of t's index set.

As a notational convention, in practice we will often use a leaf s synonymously with its label $\hat{s}$ whenever the distinction is not vital.

Lemma 4.2.1 (Properties of cluster trees). *Let $T_{\mathcal{I}}$ be a cluster tree. Then the following statements hold:*

1. *for all $t, s \in T_{\mathcal{I}}$ with $t \neq s$ and level$(t) =$ level(s), we have $\hat{s} \cap \hat{t} = \emptyset$;*
2. *for all $t, s \in T_{\mathcal{I}}$ with level$(t) \leq$ level(s) and $\hat{t} \cap \hat{s} \neq \emptyset$, we have $s \in$ sons$^*(t)$;*
3. *for all $i \in \mathcal{I}$, there is a leaf $t \in \mathcal{L}(T_{\mathcal{I}})$ with $i \in \hat{t}$;*
4. *$\mathcal{I} = \dot{\bigcup}_{t \in \mathcal{L}(T_{\mathcal{I}})} \hat{t}$.*

Proof. For a proof, see lemma 2.2 in section 2.1 of [4].

We now consider ways to construct such cluster trees. Let us introduce a very simple framework in which several clustering strategies can be expressed.

Procedure `subdivide_cluster`(t, $c_{\max}$)

Input: a cluster t, a maximum cluster size $c_{\max}$
if $\#\hat{t} > c_{\max}$ **then**
  `children` := `split`($\hat{t}$) ;
  foreach $I \in$ `children` **do**
    add new cluster s with indices I as a son to t ;
    recursively invoke `subdivide_cluster`($s, c_{\max}$) ;
  end
end

This procedure is initially invoked on a trivial tree consisting only of a root node which comprises the full index set $\mathcal{I}$. It then recursively splits nodes until each leaf node contains at most $c_{\max}$ indices. The actual details of how to split a cluster into two or more children are abstracted into the procedure `split`, which returns a partition of the index set it is passed.

There are several strategies available for performing this; in the following, we will present two such strategies.

First, some preliminaries. A splitting strategy has to take the relative positions of the basis function supports $(\mathrm{supp}(\phi_i))_{i\in\mathcal{I}}$ into consideration. In practice, however, working with the exact support sets is too complicated to be feasible, and we therefore choose a representative $x_i \in \mathrm{supp}(\phi_i)$ for each index $i \in \mathcal{I}$. Since our basis functions $(\phi_i)_{i\in\mathcal{I}}$ are assumed to have local support, this is a reasonable simplification. A possible choice for x_i is e.g. the center of mass of the basis support.

Axis-aligned bounding box splitting

Given an index set $I \subset \mathcal{I}$, determine the extents of an axis-aligned bounding box,

$$a_l := \min\{(x_i)_l : i \in I\}, \quad b_l := \max\{(x_i)_l : i \in I\} \quad \text{for } 1 \le l \le d.$$

Thus, $x_i \in A$ for all $i \in I$ with the box $A := [a_1, b_1] \times \ldots \times [a_d, b_d]$. Now determine the axis along which A has its maximum extent,

$$k := \operatorname*{arg\,max}_{1\le l\le d}(b_l - a_l),$$

and split A into two subsets along this axis:

$$A_1 := \{x \in A : x_k \le \frac{b_k - a_k}{2}\}, \quad A_2 := A \setminus A_1.$$

The partition of the index set I is then given by

$$I_1 := \{i \in I : x_i \in A_1\}, \quad I_2 := \{i \in I : x_i \in A_2\}.$$

As a variation, it is possible to split the box A along every axis simultaneously which yields a partition of 2^d subsets instead of 2 as in the original variant.

Principal component analysis

Given an index set $I \subset \mathcal{I}$, we first compute the center of mass of the representatives,

$$\bar{x} := \frac{1}{\#I}\sum_{i\in I} x_i,$$

and the (symmetric) covariance matrix $\Sigma \in \mathbb{R}^{d\times d}$ with the entries

$$\Sigma_{jk} = \frac{1}{\#I}\sum_{i\in I}((x_i)_j - \bar{x}_j)((x_i)_k - \bar{x}_k).$$

One then computes the eigenvector v_1 belonging to the largest (in magnitude) eigenvalue of Σ by a von Mises iteration. This vector points in the direction of the *principal component* of the given cluster. A partition of the index set I is then constructed by putting elements of I into either one of two subsets I_1 and I_2 according to on which side of the plane through $\bar{x}$ with normal vector v_1 they lie. Expressed symbolically, we get

$$I_1 := \{i \in I : (x_i - \bar{x}) \cdot v_1 \leq 0\}, \quad I_2 := I \setminus I_1.$$

Similar to the previous method, this strategy splits the given cluster along its "longest" component, but in a fashion that is not constrained to the coordinate axes and that is weighted with respect to the positions of the representatives (x_i).

4.2.2 Block cluster trees

Let us assume that we have some kernel function

$$K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$$

which we want to discretize with the help of two (possibly identical) families of basis functions,

$$(\phi_i)_{i \in \mathcal{I}}, \qquad (\psi_j)_{j \in \mathcal{J}}$$

with index sets $\mathcal{I}$ and $\mathcal{J}$. As in the previous section, we assume local support for the basis functions ϕ_i and ψ_j, but leave them otherwise unspecified. Assuming a Galerkin discretization, we obtain a matrix $A \in \mathbb{R}^{\mathcal{I} \times \mathcal{J}}$ with the entries

$$A_{ij} = \int_{\mathrm{supp}(\phi_i)} \int_{\mathrm{supp}(\phi_j)} \phi_i(x) K(x,y) \psi_j(y) \, ds_y \, ds_x, \quad i \in \mathcal{I}, j \in \mathcal{J}.$$

Remark. Note that we use a slightly more general notion of a matrix here than is common: we define a matrix $A \in \mathbb{R}^{\mathcal{I} \times \mathcal{J}}$ to be a mapping

$$A : \mathcal{I} \times \mathcal{J} \to \mathbb{R},$$

but do not require the row and column index sets $\mathcal{I}$ and $\mathcal{J}$ to be contiguous sets of integers starting with 1. This simplifies the notation for sub-blocks of such a matrix A.

Let us now consider a sequence of sub-matrix blocks

$$A_k \in \mathbb{R}^{s_k \times t_k}, \quad 1 \leq k \leq N_B.$$

Each block A_k is constructed by choosing subsets of the index sets,

$$s_k \subset \mathcal{I}, \; t_k \subset \mathcal{J} \quad \text{for } 1 \leq k \leq N_B,$$

and cropping A so as to contain only the entries which lie on any of the rows indicated by s_k and on any of the columns indicated by t_k. Using the notion of a matrix as a mapping mentioned above, we may simply write

$$A_k := A|_{s_k \times t_k}.$$

The set

$$\{s_k \times t_k : \ 1 \leq k \leq N_B\}$$

is called the set of *block clusters*. In order for the sequence A_k to be a partition of the matrix A, the following condition obviously has to hold:

$$\mathcal{I} \times \mathcal{J} = \dot{\bigcup}_{k \in \{1,\ldots,N_B\}} s_k \times t_k.$$

We encode these requirements in the following definition.

Definition 4.2.5 (Block cluster trees). Let $T_\mathcal{I}$ and $T_\mathcal{J}$ be cluster trees for $\mathcal{I}$ and $\mathcal{J}$, respectively. A labeled tree $T_{\mathcal{I}\times\mathcal{J}}$ with the label set $\mathcal{P}(\mathcal{I} \times \mathcal{J})$ is called *block cluster tree* for $T_\mathcal{I}$ and $T_\mathcal{I}$ if and only if

- $\operatorname{root}(T_{\mathcal{I}\times\mathcal{J}}) = (\operatorname{root}(T_\mathcal{I}), \operatorname{root}(T_\mathcal{J}))$;
- each node $b \in T_{\mathcal{I}\times\mathcal{J}}$ has the form $b = (s, t)$ with clusters $s \in T_\mathcal{I}$ and $t \in T_\mathcal{J}$;
- for each node $b = (s, t) \in T_{\mathcal{I}\times\mathcal{J}}$ with $\operatorname{sons}(b) \neq \emptyset$, we have

$$\operatorname{sons}(b) = \begin{cases} \{(s, t') : t' \in \operatorname{sons}(t)\}, & \operatorname{sons}(s) = \emptyset \wedge \operatorname{sons}(t) \neq \emptyset, \\ \{(s', t) : s' \in \operatorname{sons}(s)\}, & \operatorname{sons}(s) \neq \emptyset \wedge \operatorname{sons}(t) = \emptyset, \\ \{(s', t') : s' \in \operatorname{sons}(s), t' \in \operatorname{sons}(t)\}, & \text{else;} \end{cases}$$

- the label of a node $b = (s, t) \in T_{\mathcal{I}\times\mathcal{J}}$ is given by $\hat{b} = \hat{s} \times \hat{t} \subset \mathcal{I} \times \mathcal{J}$.

Note that a block cluster tree $T_{\mathcal{I}\times\mathcal{J}}$ is in fact a special cluster tree for $\mathcal{I}\times\mathcal{J}$ where the splitting of nodes may only occur along the rows and columns of the matrix A, so to speak. This is required so that A is partitioned into the sequence A_k of rectangular submatrices. The block clusters (s_k, t_k) used to represent A hierarchically are thus given by the labels of the leaves of the block cluster tree $T_{\mathcal{I}\times\mathcal{J}}$.

Note that, for given cluster trees $T_\mathcal{I}$ and $T_\mathcal{J}$, definition 4.2.5 does not leave much freedom for constructing a block cluster tree $T_{\mathcal{I}\times\mathcal{J}}$. If we start with the prescribed root node and apply a recursive subdivision algorithm as we did for cluster trees (cf. procedure `subdivide_cluster` on page 41), then in fact the only choice left is when to stop the subdivision. This choice is influenced by the vaguely formulated requirements that

- as many of the sub-matrix blocks A_k as possible should be well-approximable by a low-rank matrix,
- the approximable sub-matrix blocks should be as large as possible,
- the non-approximable sub-matrix blocks should be as small as possible.

This leads to the question of how to determine whether some matrix block A_k can be "well" approximated or not. Clearly, the naive approach of generating both the full matrix block and its low-rank approximation and calculating the relative error of the approximation is infeasible for reasons of computational effort. Hence, we will introduce a heuristic which allows us to assess the quality of the approximation *a priori*. For this, we assume that the kernel $K(x, y)$ has a singularity for $x = y$ (which is indeed the case for all three boundary integral kernels we have encountered; cf. eqs. (3.16)–(3.18)). Looking back at the example presented in section 4.1, we may conjecture that the well-approximable block clusters are those for which there is a certain amount of separation between the support of the row basis functions and the column basis functions.

Let us denote the support of a given cluster $s \in T_{\mathcal{I}}$ with associated basis functions $(\phi_i)_{i\in\mathcal{I}}$ by

$$\Omega_{\mathcal{I},s} := \bigcup_{i\in\hat{s}} \operatorname{supp}(\phi_i).$$

Definition 4.2.6 (Admissibility). We call a block cluster (s, t) with $s \in T_{\mathcal{I}}$ and $t \in T_{\mathcal{J}}$ *admissible* with parameter $\eta > 0$ if and only if

$$\min\{\operatorname{diam}(\Omega_{\mathcal{I},s}), \operatorname{diam}(\Omega_{\mathcal{J},t})\} \leq \eta \operatorname{dist}(\Omega_{\mathcal{I},s}, \Omega_{\mathcal{J},t})$$

where $\operatorname{diam}(X)$ refers to the Euclidean diameter of a set $X \subset \mathbb{R}^d$ and $\operatorname{dist}(X, Y)$ refers to the Euclidean distance between two sets $X, Y \subset \mathbb{R}^d$.

Remark. Calculating the diameter of a cluster and the distance between clusters exactly may be difficult in practice. We therefore introduce bounding boxes for all clusters, that is, axis-parallel boxes $Q_{\mathcal{I},s}$ and $Q_{\mathcal{J},t}$ with the property

$$\Omega_{\mathcal{I},s} \subseteq Q_{\mathcal{I},s}, \quad \Omega_{\mathcal{J},t} \subseteq Q_{\mathcal{J},t} \quad \forall s \in T_{\mathcal{I}}, t \in T_{\mathcal{J}}.$$

We may then replace the admissibility condition by the stronger condition

$$\min\{\operatorname{diam}(Q_{\mathcal{I},s}), \operatorname{diam}(Q_{\mathcal{J},t})\} \leq \eta \operatorname{dist}(Q_{\mathcal{I},s}, Q_{\mathcal{J},t}).$$

With the concept of admissibility as a good indicator of the approximability of a given block cluster, we may now formulate a straightforward algorithm for the construction of a block cluster tree.

Procedure `subdivide_blockcluster`$((s,t),\ \eta)$

Input: a block cluster (s,t), an admissibility parameter $\eta > 0$
if (s,t) *not* η*-admissible* $\wedge\neg(s$ *leaf* $\wedge\ t$ *leaf)* **then**
  if s *leaf* **then**
    `children` $:= \{s\} \times \mathrm{sons}(t)$;
  else if t *leaf* **then**
    `children` $:= \mathrm{sons}(s) \times \{t\}$;
  else
    `children` $:= \mathrm{sons}(s) \times \mathrm{sons}(t)$;
  end
  foreach $(s',t') \in$ `children` **do**
    add new block cluster (s',t') as a son to (s,t) ;
    recursively invoke `subdidive_blockcluster`$((s',t'),\eta)$;
  end
end

As with procedure `subdivide_cluster`, this algorithm works recursively and is initially invoked on the trivial block cluster tree consisting only of the root node $(\mathrm{root}(T_{\mathcal{I}}), \mathrm{root}(T_{\mathcal{J}}))$. For each block cluster (s,t), we first check whether it is already admissible. If so, we stop the subdivision in order to satisfy the requirement that well-approximable sub-matrix blocks should be as large as possible. Otherwise, and if at least one of the clusters s and t can be further split, we subdivide the block cluster into a number of child block clusters according to the cross product of the sons of s and t. The hope is that one or more of these child block clusters is admissible, resulting in a decrease in the size of the non-approximable matrix blocks.

With this algorithm in hand, we can now define a hierarchical matrix approximation for A. We first define

$$\tilde{A}_k := \begin{cases} \texttt{lrapprox}(A_k), & \text{if } (s_k,t_k) \text{ is admissible,} \\ A_k, & \text{else,} \end{cases}$$

where `lrapprox`(A_k) represents a suitable low-rank approximation for the admissible sub-matrix block A_k. We have already seen one such method, the truncated SVD, in section 4.1, and discuss this issue further in section 4.3. Thus, as we have mentioned before, admissible blocks of A will be approximated whereas inadmissible blocks will simply be stored as-is. The hierarchical approximation $\tilde{A}$ of A is then simply given by

$$(\tilde{A})_{ij} = (\tilde{A}_k)_{ij} \quad \text{for } k \text{ such that } (i,j) \in s_k \times t_k.$$

Example 4.2.1. We choose a unit hemisphere as our geometry and discretize it into a triangular quasi-uniform mesh consisting of 923 triangles. We choose piecewise constant ansatz functions on these triangles for both

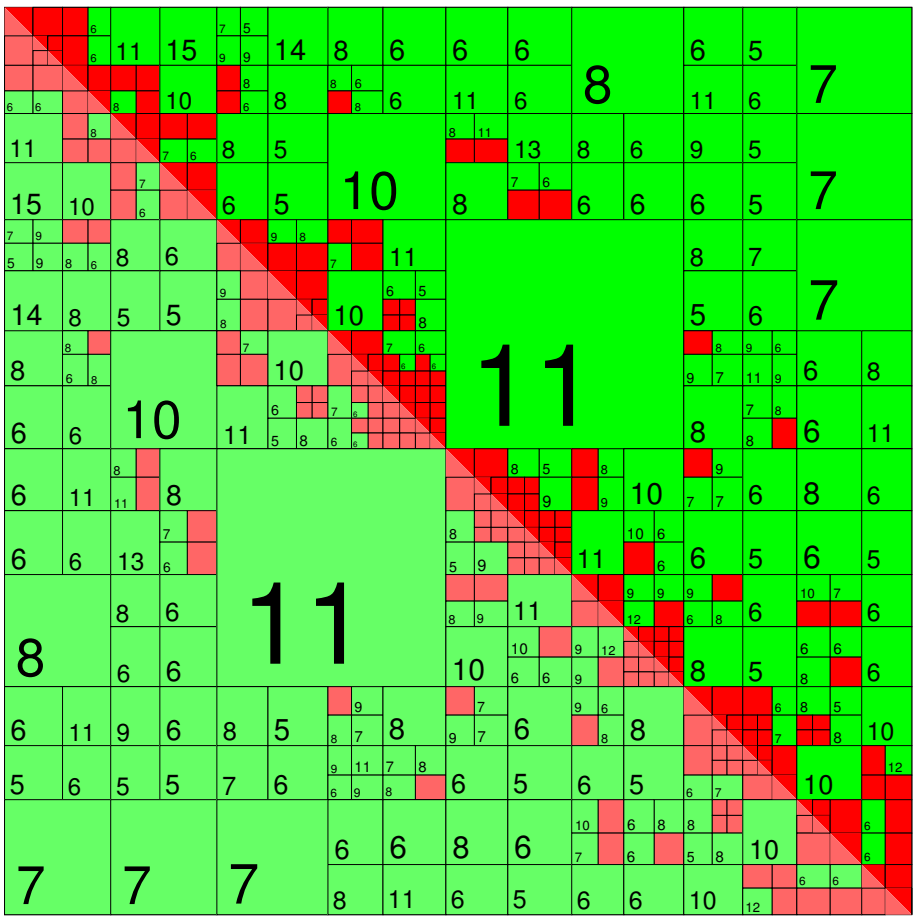

Figure 4.6: Hierarchical subdivision of the single layer potential matrix for a hemisphere.

$(\phi_i)_{i\in\mathcal{I}}$ and $(\psi_j)_{j\in\mathcal{J}}$ and create a cluster tree $T_\mathcal{I} = T_\mathcal{J}$ by applying the principal component analysis splitting strategy with a maximum cluster size $c_{\max} = 15$. This cluster tree is then supplied to the block cluster tree algorithm presented above; for the admissibility parameter, we choose $\eta = 1.1$. The resulting block cluster tree is used to generate a hierarchical matrix approximation $\tilde{V}_h$ to the single layer potential matrix V_h as given in (3.16). The structure of $\tilde{V}_h$ is represented graphically in fig. 4.6, where red blocks are inadmissible and green blocks are admissible. The numbers in the green blocks represent the rank of the low-rank approximation obtained for the admissible blocks. Storing the full matrix would require about 3.25 Mb of memory, while the hierarchical approximation requires only 1.18 Mb. This corresponds to a reduction in memory usage to approximately 38% of the original.

4.3 Low-rank matrix approximation

We will now concern ourselves with the question of how to generate a low-rank approximation $\tilde{A}_k \in \mathbb{R}^{s_k\times t_k}$ to some given matrix $A_k \in \mathbb{R}^{s_k\times t_k}$. For ease of notation, we will from now on denote the index sets by s and t and write $M \in \mathbb{R}^{s\times t}$ and $\tilde{M} \in \mathbb{R}^{s\times t}$ for the matrix and its approximation, respectively.

In section 4.1, we have discussed one possible method for generating a low-rank approximation, namely, the truncated singular value decomposition. While this method yields very good results and is even optimal in the sense of the error in the Frobenius norm, calculating the singular value decomposition of a matrix $M \in \mathbb{R}^{s\times s}$ numerically requires $\mathcal{O}\left(\#s^3\right)$ floating point operations and is thus rather slow. This is mitigated by the fact that, for hierarchical approximation, low-rank approximation is only performed on the sub-matrix blocks $(A_k)_{1\leq k\leq N_B}$ of the full matrix A: these blocks are typically much smaller than A itself. Still, for our goal of obtaining quasi-optimal performance for common matrix operations, an algorithm with lower time complexity would be desirable. Ideally, we would like an algorithm which operates in $\mathcal{O}\left(\#s\right)$ operations, i. e. linear in the number of unknowns, and produces similarly good results as the TSVD.

Our investigations will be based on the framework of *skeletal* or *cross approximation* ([7]). The idea is to choose small sets $\tilde{s} \subset s$ and $\tilde{t} \subset t$ of pivot rows and pivot columns, respectively, and represent the approximation as a matrix product

$$\tilde{M} = M|_{s\times\tilde{t}} \cdot S \cdot M|_{\tilde{s}\times t}$$

with a coefficient matrix $S \in \mathbb{R}^{\tilde{t}\times\tilde{s}}$. This may be viewed as constructing $\tilde{M}$ as a linear combination of all possible rank one outer products of the pivot rows and columns, i. e.

$$\tilde{M} \in \operatorname{span}\left\{M|_{s\times\{j\}} \cdot M|_{\{i\}\times t} : (i,j) \in \tilde{s}\times\tilde{t}\right\}.$$

Every such rank one product $M|_{s\times\{j\}} \cdot M|_{\{i\}\times t} \in \mathbb{R}^{s\times t}$ is called a *cross*; we shall denote it by $C(M, i, j)$.

The following result states that for any low-rank approximation, there is a cross approximation of similar quality.

Theorem 4.3.1 (Existence of cross approximations). *Let $M, R \in \mathbb{R}^{s\times t}$ be matrices with $\|M - R\| \leq \varepsilon$ and rank $R \leq k$. Then there exist subsets $\tilde{s} \subset s$ and $\tilde{t} \subset t$ and a matrix $S \in \mathbb{R}^{\tilde{t}\times\tilde{s}}$ such that*

$$\left\|M - (M|_{s\times\tilde{t}} \cdot S \cdot M|_{\tilde{s}\times t})\right\|_2 \leq \varepsilon(1 + 2\sqrt{k}(\sqrt{\#\tilde{s}} + \sqrt{\#\tilde{t}})).$$

Proof. See [7].

We now present heuristic algorithms which can be used to compute such cross approximations.

4.3.1 Cross approximation with full pivoting

This algorithm is based on finding the entry M_{ij} of M with maximum modulus $|M_{ij}|$ and choosing i and j as pivot row and column, respectively. The cross $C(M, i, j)$ is then added to the cross approximation, and the process is repeated on the matrix $M - C(M, i, j)$.

Algorithm 3: Cross approximation with full pivoting

Input: a matrix $M \in \mathbb{R}^{s\times t}$, a maximum rank $k \in \mathbb{N}$
Output: low-rank approximation $R_k = \sum_{\nu=1}^{k} a^\nu (b^\nu)^T$
for $\nu = 1, \ldots, k$ **do**
  $(i_\nu, j_\nu) := \arg\max_{(i,j)} |M_{ij}|$;
  $\delta := M_{i_\nu j_\nu}$;
  if $\delta = 0$ **then**
    terminate with exact rank $\nu - 1$ representation $R_{\nu-1}$ of M ;
  else
    pivot column vector $(a^\nu)_i := M_{ij_\nu}$ for $i \in s$;
    pivot row vector $(b^\nu)_j := \frac{1}{\delta} M_{i_\nu j}$ for $j \in t$;
    $M_{ij} := M_{ij} - (a^\nu)_i (b^\nu)_j$ for $(i, j) \in s \times t$;
  end
end

This algorithm is easy to implement and has the following interesting properties.

Lemma 4.3.2. *Let $M \in \mathbb{R}^{s\times t}$ be a matrix with rank $M = k$. Then algorithm 3 terminates after exactly k iterations and finds an exact representation*

$$\sum_{\nu=1}^{k} a^\nu (b^\nu)^T = M.$$

Proof. See [4].

Lemma 4.3.3. *Let $M \in \mathbb{R}^{s\times t}$ be a matrix with rank $M \geq 1$ and let R_k be the cross approximation computed by algorithm 3. Then for any pivot row i^ν or column j^ν, $1 \leq \nu \leq k$, we have*

$$R_k|_{s\times\{j^\nu\}} = M|_{s\times\{j^\nu\}},$$
$$R_k|_{\{i^\nu\}\times t} = M|_{\{i^\nu\}\times t}.$$

In other words, R_k reproduces all pivot rows and pivot columns of M exactly.

Proof. See [4].

Algorithm 3 however still suffers from quadratic runtime complexity; more precisely, it requires $\mathcal{O}(k\,\#s\,\#t)$ operations. Next we consider an algorithm which does not suffer from this problem.

4.3.2 Cross approximation with partial pivoting

In every iteration $\nu = 1, \ldots, k$ of algorithm 3, we have to

- determine the pivot point (i_ν, j_ν) ($\mathcal{O}(\#s\,\#t)$ operations),
- compute the vectors a^ν and b^ν ($\mathcal{O}(\#s + \#t)$ operations) and
- update the matrix M ($\mathcal{O}(\#s\,\#t)$ operations).

Clearly, the first and third steps are the bottlenecks bringing the algorithm to quadratic complexity. Note that even having to precompute all entries of the matrix M before the algorithm starts makes the algorithm inherently quadratic, but since we access every entry of M in every iteration, this requirement can hardly be eliminated in algorithm 3.

The third step above can easily be reduced to linear complexity by storing not the complete residuum $M - R_k$, but instead only the vectors a^ν and b^ν and computing the entries $R_k = \sum_{\nu=1}^{k} a^\nu (b^\nu)^T$ as needed on the fly. The remaining question, then, is how to choose pivot points in an efficient manner.

Here we introduce the concept of partial pivoting whereby the modulus $|M_{ij}|$ is not maximized over both i and j, but instead one index is held fixed. This corresponds to finding the maximum modulus only along one row or column instead of over the entire matrix. For any given row index $i^* \in s$, we need $\mathcal{O}(\#t)$ operations to compute the corresponding matrix row and the row maximizer

$$j^* = \arg\max_{j\in t} |M_{i^*j}|\,.$$

The algorithm in full is given below.

Algorithm 4: Cross approximation with partial pivoting

Input: a matrix $M \in \mathbb{R}^{s \times t}$, a maximum rank $k \in \mathbb{N}$
Output: low-rank approximation $R_k = \sum_{\nu=1}^{k} a^\nu (b^\nu)^T$
set $i^* := \min(s), \nu := 1, \mathcal{P} := \emptyset$;
while $\nu \leq k$ **do**
  $j^* := \arg\max_{j \in t} |M_{ij}|$;
  $\delta := M_{i^* j^*} - \sum_{\mu=1}^{\nu-1} (a^\mu)_{i^*} (b^\mu)_{j^*}$;
  if $\delta = 0$ **then**
    if $\#\mathcal{P} = n - 1$ **then**
      terminate with exact rank $\nu - 1$ representation $R_{\nu-1}$ of M ;
    end
  else
    $(a^\nu)_i := M_{ij^*} - \sum_{\mu=1}^{\nu-1} (a^\mu)_i (b^\mu)_{j^*}$ for $i \in s$;
    $(b^\nu)_j := \frac{1}{\delta} \left(M_{i^* j} - \sum_{\mu=1}^{\nu-1} (a^\mu)_{i^*} (b^\mu)_j \right)$ for $j \in t$;
    $\nu := \nu + 1$;
  end
  $\mathcal{P} := \mathcal{P} \cup \{i^*\}$;
  choose $i^* \in s \setminus \mathcal{P}$, e.g. $i^* = \arg\max_{i \in s \setminus \mathcal{P}} |(b^\nu)_{ij^*}|$;
end

If we can guarantee $\delta \neq 0$ at every iteration, then algorithm 4 completes after $\mathcal{O}\left(k^2(\#s + \#t)\right)$ operations and the statement of lemma 4.3.2 remains valid. For sparse matrices, finding a row with $\delta \neq 0$ may take several iterations which can bring the algorithm to quadratic complexity. For our dense boundary element matrices, however, the method may be well-suited.

4.3.3 Adaptive Cross Approximation (ACA)

In both algorithm 3 and algorithm 4, we have always specified a predetermined rank k which should be used for the approximation of the matrix M. One might wish for a variant of these algorithms where only a desired accuracy ε has to be specified; the algorithm should terminate once this accuracy has been achieved, thus not using a higher rank than necessary. For this, some method for estimating the approximation error $\|M - R_k\|$ (or the corresponding relative error) for a given k is required. Clearly, doing this exactly is bound to make the cross approximation at least quadratic in runtime complexity and should thus be avoided if possible.

We therefore use the heuristic of estimating the error $\|M - R_k\|$ by a rank one approximation,

$$\|M - R_k\| \lesssim \|M - R_{k-1}\| \approx \|R_k - R_{k-1}\| = \left\| a^k (b^k)^T \right\| = \left\| a^k \right\|_2 \left\| b^k \right\|_2$$

(which is valid in both the spectral and the Frobenius norm). Hence we may

estimate the absolute and relative errors in the Frobenius norm by

$$\epsilon^F_{\mathrm{abs}}(k) := \left\|a^k\right\|_2 \left\|b^k\right\|_2,$$

$$\epsilon^F_{\mathrm{rel}}(k) := \frac{\left\|a^k\right\|_2 \left\|b^k\right\|_2}{\sqrt{\sum_{\nu=1}^k \|a^\nu\|_2^2 \|b^\nu\|_2^2 + 2\sum_{1\leq\mu<\nu\leq k} (a^\mu)^T a^\nu (b^\mu)^T b^\nu}}$$

where the denominator of $\epsilon^F_{\mathrm{rel}}(k)$ is the Frobenius norm of $R_k = \sum_{\nu=1}^k a^\nu (b^\nu)^T$ and serves as an approximation of $\|M\|_F$.

The criterion "$\nu \leq k$" in algorithm 3 or algorithm 4 which determines whether another iteration should be performed is then replaced by

$$\epsilon(\nu - 1) > \varepsilon$$

where ϵ is either of the functions $\epsilon^F_{\mathrm{abs}}$ or $\epsilon^F_{\mathrm{rel}}$, and we set $\epsilon(0) := \infty$. Because the rank is determined adaptively, these variants are called *adaptive cross approximation* (ACA).

Convergence cannot always be guaranteed for this formulation of the algorithm. In particular, Example 4.9 in [4] demonstrates that for certain configurations of clusters on geometry with discontinuous normals, we can have $\epsilon \to 0$ while $\|M - R_k\| = \mathcal{O}(1)$ when generating the double layer potential matrix K_h. Thus, we cannot bound the error in the matrix block being generated and the low-rank approximation fails to give meaningful results.

Therefore, a modification of the algorithm is required in order to deal with these situations. A simple heuristic which seems to work reliably in practice is to keep track of the number of successful approximations performed on each individual row and prefer rows with smaller counts as pivot rows. The details are described in [2].

4.4 Operations on hierarchical matrices

By applying the methods described in the previous sections, we ultimately obtain a hierarchical matrix $\tilde{A} \in \mathbb{R}^{\mathcal{I}\times\mathcal{J}}$ with

$$(\tilde{A})_{ij} = (\tilde{A}_k)_{ij} \quad \text{for } k \text{ such that } (i,j) \in s_k \times t_k$$

where (s_k, t_k) for $1 \leq k \leq N_B$ are the leaves of a block cluster tree $T_{\mathcal{I}\times\mathcal{J}}$. Each block $\tilde{A}_k \in \mathbb{R}^{s_k\times t_k}$ is stored in low-rank form if (s_k, t_k) is admissible and in standard dense matrix form otherwise. Formally, we denote the set of hierarchical matrices by

$$\mathcal{H}(T_{\mathcal{I}\times\mathcal{J}}, r) := \{M \in \mathbb{R}^{\mathcal{I}\times\mathcal{J}} : \operatorname{rank} M|_{s\times t} \leq r \text{ for all admissible leaves } (s,t) \text{ of } T_{\mathcal{I}\times\mathcal{J}}\}.$$

In the following, we give several operations on a hierchical matrices in this form.

4.4.1 Matrix truncation

We first define truncation for individual matrix blocks.

Definition 4.4.1 (Truncation). Let $M \in \mathbb{R}^{m\times n}$. For a fixed $r \in \mathbb{N}$, we define the truncation operator

$$\begin{aligned}\mathcal{T}_r : \mathbb{R}^{m\times n} &\to \mathcal{R}(r,m,n)\\ M &\mapsto \tilde{M} = \operatorname*{arg\,min}_{X\in\mathcal{R}(r,m,n)} \|X - M\|.\end{aligned}$$

We might also wish to perform truncation to a predetermined accuracy without fixing the approximation rank r.

Definition 4.4.2 (Adaptive truncation). Let $M \in \mathbb{R}^{m\times n}$. For a fixed $\varepsilon > 0$, we define the adaptive truncation operator

$$\mathcal{T}_\varepsilon(M) := \mathcal{T}_r(M) \text{ where}$$
$$r = \min\left\{\tilde{r} \in \mathbb{N}_0 \mid \exists \tilde{M} \in \mathcal{R}(\tilde{r},m,n) : \left\|M - \tilde{M}\right\| \leq \varepsilon \|M\|\right\}.$$

We now extend the definition of truncation to hierarchical matrices.

Definition 4.4.3 (Hierarchical truncation). Let $M \in \mathbb{R}^{\mathcal{I}\times\mathcal{J}}$ be a matrix and $T_{\mathcal{I}\times\mathcal{J}}$ a block cluster tree for the index sets $\mathcal{I}$ and $\mathcal{J}$. For any $r \in \mathbb{N}$, we define the hierarchical truncation operator

$$\begin{aligned}\mathcal{T}_r : \mathbb{R}^{\mathcal{I}\times\mathcal{J}} &\to \mathcal{H}(T_{\mathcal{I}\times\mathcal{J}}, r)\\ M &\mapsto \tilde{M}\end{aligned}$$

where for all leaves $(s,t) \in T_{\mathcal{I}\times\mathcal{J}}$, we have

$$\tilde{M}|_{s\times t} = \begin{cases}\mathcal{T}_r(M|_{s\times t}), & \text{if } (s,t) \text{ admissible},\\ M|_{s\times t} & \text{otherwise.}\end{cases}$$

The adaptive hierarchical truncation $\mathcal{T}_\varepsilon(M)$ is defined analogously.

4.4.2 Matrix addition and multiplication

Given two hierarchical matrices $A, B \in \mathcal{H}(T_{\mathcal{I}\times\mathcal{J}}, r)$, in general we have $A + B \in \mathcal{H}(T_{\mathcal{I}\times\mathcal{J}}, 2r)$. Hence, in order to keep the rank constant, we define the *formatted addition* of $\mathcal{H}$-matrices by

$$A \oplus B := \mathcal{T}_r(A + B).$$

The product AB for matrices $A, B \in \mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, r)$ has a more complicated structure than the addition (where simply corresponding blocks of A and B

are added), and again the result will in general not lie in $\mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, r)$. We therefore define the *formatted multiplication* as

$$A \otimes B := \mathcal{T}_r(AB).$$

Note that calculating the best truncated representation $\mathcal{T}_r$ is usually too expensive in practice, and therefore approximations are used. The details are quite involved and beyond the scope of this work, and we refer the reader to [4] for reference.

4.4.3 Matrix-vector multiplication

Given a hierarchical matrix $A \in \mathcal{H}(T_{\mathcal{I}\times\mathcal{J}}, r)$ with blocks $(A_k)_{1\leq k\leq N_B}$ and a vector $x \in \mathbb{R}^{\mathcal{J}}$, we wish to compute the product $Ax \in \mathbb{R}^{\mathcal{I}}$. We denote the restriction of the vector x to the cluster t_k by $x|_{t_k} \in \mathbb{R}^{t_k}$, and the extension of a vector $y_k \in \mathbb{R}^{s_k}$ to s by

$$(y_k|^s)_i = \begin{cases} (y_k)_i, & i \in s_k, \\ 0, & \text{otherwise} \end{cases} \quad \text{for } i \in \mathcal{I}.$$

It is easy to see that the product Ax can then be computed by

$$Ax = \sum_{k=1}^{N_B} (A_k x|_{t_k})|^s.$$

For inadmissible blocks (s_k, t_k), the multiplication $A_k x|_{t_k}$ is a standard dense matrix-vector multiplication. For admissible blocks, we have the low-rank representation

$$A_k = \sum_{\nu=1}^{r_k} a_k^\nu (b_k^\nu)^T$$

with vectors $a_k^\nu \in \mathbb{R}^{s_k}$ and $b_k^\nu \in \mathbb{R}^{t_k}$ for every $\nu \in \{1, \ldots, r_k\}$ and hence

$$(A_k x|_{t_k})_i = \sum_{\nu=1}^{r_k} (a_k^\nu)_i \sum_{j\in t_k} (b_k^\nu)_j x_j, \quad \forall i \in s_k.$$

Computing the product $A_k x|_{t_k}$ thus requires $\mathcal{O}\left(r_k \#s_k \#t_k\right)$ operations.

4.4.4 Solving a triangular system

Given a lower triangular hierarchical matrix $L \in \mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, r)$, we may wish to solve a system

$$Lx = y$$

with given $y \in \mathbb{R}^{\mathcal{I}}$ for $x \in \mathbb{R}^{\mathcal{I}}$. This is accomplished recursively:

- If L is not subdivided, solve $Lx = y$ by a standard dense triangular solver.
- If L is subdivided, say for simplicity into two by two blocks

$$L = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix},$$

  solve (recursively) first

$$L_{11}x_1 = y_1$$

  and then

$$L_{22}x_2 = y_1 - L_{21}x_1.$$

This scheme may be generalized to handle matrix equations

$$LX = Y$$

with $X, Y \in \mathbb{R}^{\mathcal{I}\times\mathcal{J}}$. Upper triangular systems are of course treated analogously.

4.4.5 Cholesky and LU factorization

In a similar way, we may compute an LU factorization of a hierarchical matrix $A \in \mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, r)$ by a recursive procedure:

- If A is not subdivided, compute $LU = A$ by a standard dense LU factorization routine.
- If A is subdivided, say for simplicity into two by two blocks

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix},$$

  then we (recursively) compute the factorization

$$L_{11}U_{11} = A_{11},$$

  solve the triangular systems

$$L_{11}U_{12} = A_{12} \quad \text{and} \quad L_{21}U_{11} = A_{21}$$

  for U_{12} and L_{21} as described in section 4.4.4 and finally compute again a factorization

$$L_{22}U_{22} = A_{22} - L_{21}U_{12}.$$

  We thus obtain all components of the factors

$$L = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \quad \text{and } U = \begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix}.$$

For symmetric matrices one can compute a Cholesky factorization in an analogous way.

Note that the data-sparse matrix product $L_{21}U_{11}$ is in fact computed by the formatted multiplication $L_{21} \otimes U_{11}$ as defined in section 4.4.2. Thus, accuracy may be specified either in the form of a fixed rank k or of an adaptive error bound ε. This gives us the possibility to compute approximate matrix decompositions for use in preconditioning. We will make use of this in section 4.5.

4.5 Application to the BEM system

Consider again the BEM matrix equation (3.15),

$$\begin{pmatrix} -V_h & K_h \\ K_h^T & D_h \end{pmatrix} \begin{pmatrix} \underline{v}_{Dh} \\ \underline{u}_{Nh} \end{pmatrix} = \begin{pmatrix} -\underline{f}_{Dh} \\ \underline{f}_{Nh} \end{pmatrix} \tag{4.1}$$

with

$$\begin{aligned} V_h &\in \mathbb{R}^{N_{DT} \times N_{DT}}, \\ K_h &\in \mathbb{R}^{N_{DT} \times N_{NV}}, \\ D_h &\in \mathbb{R}^{N_{NV} \times N_{NV}}, \\ \underline{v}_{Dh}, \underline{f}_{Dh} &\in \mathbb{R}^{N_{DT}}, \\ \underline{u}_{Nh}, \underline{f}_{Nh} &\in \mathbb{R}^{N_{NV}}. \end{aligned}$$

Note that we have reversed the sign on the first equation in order to obtain a symmetric system.

Denote by $(\psi_i)_{i \in \mathcal{I}}$ with $\mathcal{I} = \{1, \ldots, N_{DT}\}$ the piecewise constant basis functions over the Dirichlet triangles, and by $(\phi_j)_{j \in \mathcal{J}}$ with $\mathcal{J} = \{1, \ldots, N_{NV}\}$ the piecewise linear ansatz functions for the Neumann vertices, as described in section 3.3. We thus have

$$V_h \in \mathbb{R}^{\mathcal{I} \times \mathcal{I}}, \quad K_h \in \mathbb{R}^{\mathcal{I} \times \mathcal{J}}, \quad D_h \in \mathbb{R}^{\mathcal{J} \times \mathcal{J}}.$$

We construct appropriate cluster trees $T_{\mathcal{I}}$ and $T_{\mathcal{J}}$ for the Dirichlet triangles and Neumann vertices, respectively, as well as block cluster trees $T_{\mathcal{I} \times \mathcal{I}}$, $T_{\mathcal{I} \times \mathcal{J}}$ and $T_{\mathcal{J} \times \mathcal{J}}$. These are then used to create hierarchical matrix approximations

$$\tilde{V}_h \in \mathcal{H}(T_{\mathcal{I} \times \mathcal{I}}, r_V), \quad \tilde{K}_h \in \mathcal{H}(T_{\mathcal{I} \times \mathcal{J}}, r_K), \quad \tilde{D}_h \in \mathcal{H}(T_{\mathcal{J} \times \mathcal{J}}, r_D).$$

While the admissibility parameter η can be chosen more or less independently of the mesh size h, matters are slightly more complicated concerning the approximation accuracy ϵ. Let us consider the approximation property

$$\inf_{x_h \in \Gamma_h} \|x - x_h\|_\Gamma \leq h^r \|v\|_\Xi \ \forall v \in \Xi$$

in some appropriate norm and with a space $\Xi \subset \Gamma$ of higher regularity. Bebendorf [2] suggests that choosing ϵ on the order of h^r is sufficient for the approximation error not to dominate the other error terms. For sufficiently smooth functions v, our basis functions achieve an approximation order of $r = 2$ as indicated by (3.11), and hence we choose $\epsilon = \mathcal{O}\left(h^2\right)$.

For the solution of the linear system, we employ an iterative solver, namely the MINRES method. The following preconditioning scheme (due to [2]) is employed. The system matrix in (4.1) has the factorization

$$\begin{pmatrix} -V_h & K_h \\ K_h^T & D_h \end{pmatrix} = \begin{pmatrix} I & 0 \\ -K_h^T V^{-1} & I \end{pmatrix} \begin{pmatrix} -V_h & K_h \\ 0 & K_h^T V_h^{-1} K_h + D_h \end{pmatrix}$$

which could already be used for preconditioning. Computing the Schur complement $K_h^T V_h^{-1} K_h + D_h$ can however be expensive even when using hierarchical matrices. It is therefore approximated by D_h which yields the preconditioner

$$C = \begin{pmatrix} I & 0 \\ -K_h^T V^{-1} & I \end{pmatrix} \begin{pmatrix} -V_h & K_h \\ 0 & D_h \end{pmatrix}.$$

For the evaluation of C^{-1}, we compute the approximate (hierarchical) Cholesky factorizations

$$\tilde{C}_V^T \tilde{C}_V \approx \tilde{V}_h,$$
$$\tilde{C}_D^T \tilde{C}_D \approx \tilde{D}_h$$

as described in section 4.4.5. As mentioned there, we may specify the desired accuracy of the decomposition by limiting the rank of the formatted multiplication used during the Cholesky factorization. For preconditioning, a rather low accuracy is sufficient in order to obtain a spectrally equivalent preconditioner without expending too much computational effort. In our numerical examples we use an adaptive accuracy of $\varepsilon = 0.05$.

We furthermore compute the matrix

$$\tilde{X} = \tilde{C}_V^{-T} \tilde{K}_h$$

by solving the matrix equation $\tilde{C}_V^T \tilde{X} = \tilde{K}_h$ as described in section 4.4.4. The preconditioner can then be evaluated by solving the system

$$\begin{pmatrix} \tilde{C}_V^T & 0 \\ -\tilde{X}^T & \tilde{C}_D^T \end{pmatrix} \begin{pmatrix} -\tilde{C}_V & \tilde{X} \\ 0 & \tilde{C}_D \end{pmatrix} \begin{pmatrix} v \\ u \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix}.$$

The system matrix itself is evaluated by computing four hierarchical matrix-vector products,

$$\begin{pmatrix} -\tilde{V}_h & \tilde{K}_h \\ \tilde{K}_h^T & \tilde{D}_h \end{pmatrix} \begin{pmatrix} v \\ u \end{pmatrix}.$$

Analogously, we compute data-sparse approximations for the matrices occurring on the right-hand side (3.19) (except the mass matrices $\bar{M}_{1h}$ and $\bar{M}_{2h}$ which are naturally sparse) in order to be able to evaluate the right-hand side terms efficiently.

Chapter 5

Numerical Results

The concepts detailed in the previous chapters were implemented in a C++ program in order to be able to perform numerical tests. There are several software libraries available which are concerned with the generation and manipulation of hierarchical matrices. Two of these were taken into consideration for integration into our program: HLib [3] by Börm and Grasedyck and AHMED [1] by Bebendorf. After a cursory inspection, we found that the sample programs included with AHMED were a more convenient starting point for our aims, and it was thus chosen as the basis of our BVP solver implementation. No detailed comparison of the two libraries was however performed, and no statement about their relative merits is implied.

5.1 Data-sparse approximation results

We first experimentally determine the approximation properties of hierarchical approximation by partially pivoted ACA as described in Chapter 4. As a model matrix, we choose the single layer potential matrix (3.16), and for the geometry, a quasi-uniform triangular approximation of the boundary of the unit sphere which is generated by tesselating a regular icosahedron.

During testing, it became apparent that the bounding box based admissibility condition underestimated distances quite dramatically in many cases, which is why a larger admissibility parameter $\eta = 5$ has been chosen. For the accuracy parameter, $\varepsilon = h^2$ was chosen as rationalized in section 4.5.

Figure 5.1 depicts the relative error of the hierarchically approximated matrix with these parameters as compared to the full matrix V_h. We see that as we increase the number of boundary elements, the approximation quality improves. More precisely, one can observe convergence of (approximately) the order of $\mathcal{O}\left(h^2\right) = \mathcal{O}\left(N_T^{-2}\right)$, i. e. linear convergence in the number of unknowns.

Next, we measure the time taken for the construction of the matrix approximation. Figure 5.2 shows the execution time for the hierarchical ap-

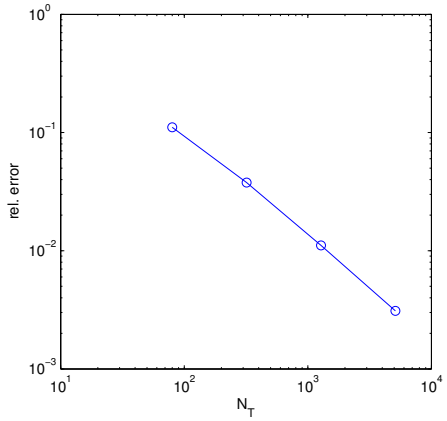

Figure 5.1: Convergence of hierarchical approximation. x-axis: number of boundary elements. y-axis: relative error in Frobenius norm.

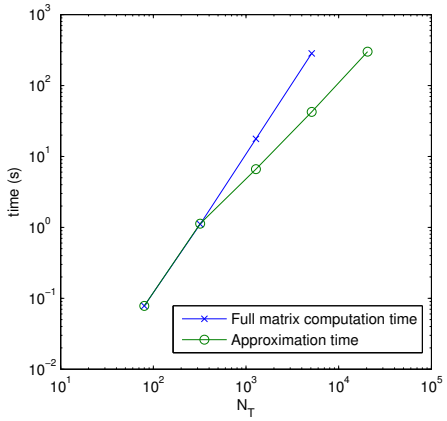

Figure 5.2: Time for hierarchical approximation. x-axis: number of boundary elements. y-axis: execution time in seconds.

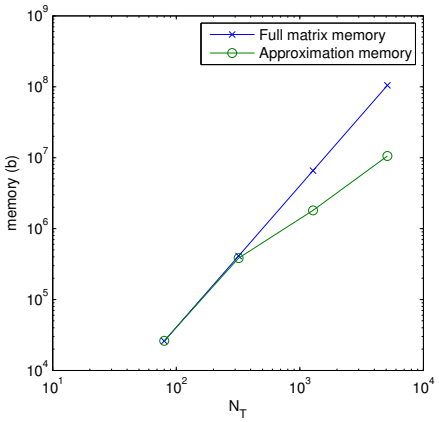

Figure 5.3: Memory usage. x-axis: number of boundary elements. y-axis: required storage in bytes.

proximation and, for reference, the time needed to construct the full matrix. The latter unsurprisingly exhibits quadratic behaviour in the number of boundary elements as $V_h \in \mathbb{R}^{N_T \times N_T}$. The times for hierarchical approximation show better asymptotic behaviour except for very small problems; their complexity corresponds to about $\mathcal{O}\left(N_T^{1.35}\right)$ for this example.

We also summarize these numbers in the following table. The second and third columns give the execution time for full and approximated computation of the matrix respectively, and the last column gives the ratio between approximation time and full time. Note that full computation of the largest matrix would have required about 1.6 GB of memory. The projected computation time would have been about 75 minutes, whereas the hierarchical approximation was computed in a mere 5 minutes.

N_T	full time (s)	approx. time (s)	ratio
80	0.078	0.078	1.00
320	1.11	1.125	1.01
1280	17.703	6.609	0.76
5120	284.031	42.531	0.15
20480	n/a	300.20	n/a

Finally, we examine the memory requirements for the full matrix and its hierarchical approximation, depicted in fig. 5.3. Here, the estimated complexity is $\mathcal{O}\left(N_T^{1.15}\right)$. We again summarize the numbers in a table.

N_T	full memory (MB)	approx. memory (MB)	ratio
80	0.0247	0.0249	1.0037
320	0.3918	0.3659	0.6948
1280	6.2549	1.7243	0.2658
5120	100.02	10.076	0.0932
20480	1600.1	61.595	0.0385

5.2 Laplace mixed BVP results

We now evaluate the performance of the BEM implementation for the mixed boundary value problem for the Laplace equation. For the geometry, we again choose the unit sphere. We prescribe Dirichlet boundary conditions on the hemisphere $\Gamma_D = \{x \in \Gamma : x_2 \geq 0\}$ and Neumann boundary conditions on its complement $\Gamma_N = \{x \in \Gamma : x_2 < 0\}$. In order to be able to compare the numerical solution to the exact one, we specify the analytic solution

$$u(x) = \frac{1}{\|x - y\|}$$

with the fixed source point $y = (-3, 2, 4)^T$. The admissibility parameter $\eta = 5$ was used for the single layer and double layer matrices, and $\eta = 2$ for the hypersingular matrix. As above, we set $\varepsilon = h^2$.

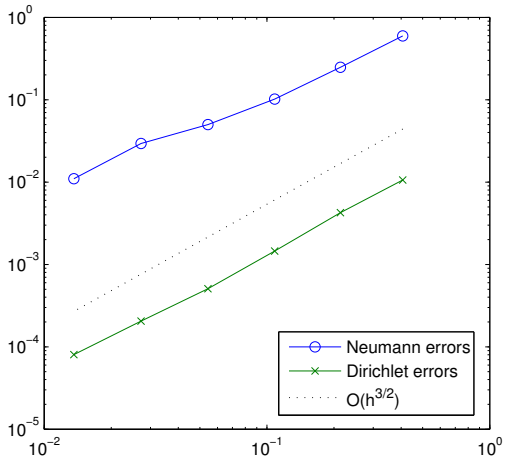

Figure 5.4: Error for unknown Dirichlet and Neumann values. x-axis: mesh size h. y-axis: relative error in L_2-norm.

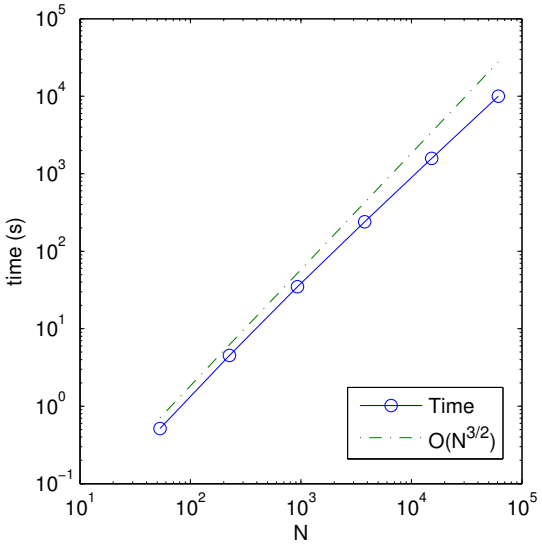

Figure 5.5: Total time for solution of BVP. x-axis: number of unknowns. y-axis: execution time in seconds.

We first examine the relative errors of the computed Dirichlet and Neumann data as compared to the exact solution. Figure 5.4 shows this information in dependence of the mesh size h. For reference, we have added a line with the behaviour $\mathcal{O}\left(h^{\frac{3}{2}}\right)$ as this seems to match the error behaviour most closely. This shows that the Dirichlet error behaves according to the error estimate (3.14) while the Neumann error exhibits better behaviour than (3.13) predicts, which may be due to additional smoothness of the solution.

We also examine the total time taken for the solution of the boundary value problem, depicted in fig. 5.5 in dependence of the number of unknowns $N = \mathcal{O}\left(h^{-2}\right)$. Specifically, we time all operations necessary for solving the problem, including construction of the data-sparse system matrices, their factorization as well as MINRES iteration. The reference line with slope $\mathcal{O}\left(N^{\frac{3}{2}}\right)$ corresponds to the number of operations a hypotethical FEM solver with perfectly linear complexity operating on $N_{FEM} = \mathcal{O}\left(h^{-3}\right)$ could achieve. For small numbers of unknowns, our implementation evidently does not perform significantly better than such a scheme. As the number of unkowns increases, though, we can see that the algorithm is able to improve upon the FEM bound, albeit not dramatically. It is conceivable that with a more finely tuned admissibility condition and approximation accuracy even better results would be possible. Bebendorf [2] gives a theoretical bound of $\mathcal{O}\left(N \log N \left|\varepsilon\right|^6\right)$ for the construction of the hierarchical matrices (which takes up the largest part of the run time by far). With our parameters, this corresponds to $\mathcal{O}\left(N(\log N)^7\right)$, although with a very large multiplicative constant.

5.3 Wave maker results

In the following, we lift a numerical example from [12] and transfer it into three dimensions. We assume that we have a test basin with the dimensions

$$\Omega = (0, 10) \times (-1, 0) \times (-1, 0).$$

The free surface at rest is the quadrilateral $\Gamma_D = (0, 10) \times (-1, 0) \times \{0\}$. On the remaining walls Γ_N, we prescribe the normal velocities

$$g_N(x, y, z, t) = \begin{cases} (1 + z)a \sin(\omega t), & x \in \{0\} \times (-1, 0) \times (-1, 0), \\ 0, & \text{otherwise.} \end{cases}$$

That is, the left basin wall is assumed to be equipped with a wave maker which exhibits periodic oscillations with maximum amplitude $a = 0.02$ and frequency $\omega = 1.8138$, and the other walls are assumed to be stationary. Note that the oscillations exhibit maximum amplitude at the top of the basin and vanish at the bottom.

The surface Γ is discretized by triangles using the software package NETGEN. In particular, we use a refined mesh with the following number of triangles and vertices:

$$11264 = N_T = N_{DT} + N_{NT} = 2560 + 8704,$$
$$5634 = N_V = N_{DV} + N_{NV} = 1377 + 4257.$$

For time discretization, we use a fixed time step $\tau = 0.1$ over the time interval $[0, 80.0]$. This results in 800 time iterations each of which requires 4 solutions of the mixed boundary value problem on the domain Ω. The entire scheme took about $7\frac{1}{2}$h to complete on a system with a single-core Athlon XP 3200+ processor and 2 GB of RAM. This results in an amortized cost of about 8.4s per solution of the BVP. Note that there is a a large cost for the generation of the system matrices which however has to be performed only once at startup. The scheme is thus better suited for long simulations where many time iterations are to be performed.

The figures on pages 65-68 depict the progression of the wave simulation. Every figure shows the wave profile over the free surface of the basin at a fixed point in time. Initially, the free surface is at rest (fig. 5.6). We then observe that the wave front induced by the wave maker begins to traverse the basin (fig. 5.7). At around $t = 28$, the wave front has reached the rear wall of the basin (fig. 5.8). It then reflects off this wall, and interference ensues between the original wave and its reflection (fig. 5.9). At about $t = 60$, the reflected wave has travelled through the entire basin back to the wave maker, and a standing wave pattern develops (fig. 5.10). Figures 5.11–5.13 show the peak, flat and inverted peak configurations of this standing wave, respectively.

We note that a slight asymmetry along the width of the basin (the y axis) develops which is not accounted for by the model. Presumably, this is an artifact of the asymmetric discretization of the basin boundary obtained via NETGEN. This deviation however apparently does not impede stability and seems to stay within the bounds of the expected numeric error.

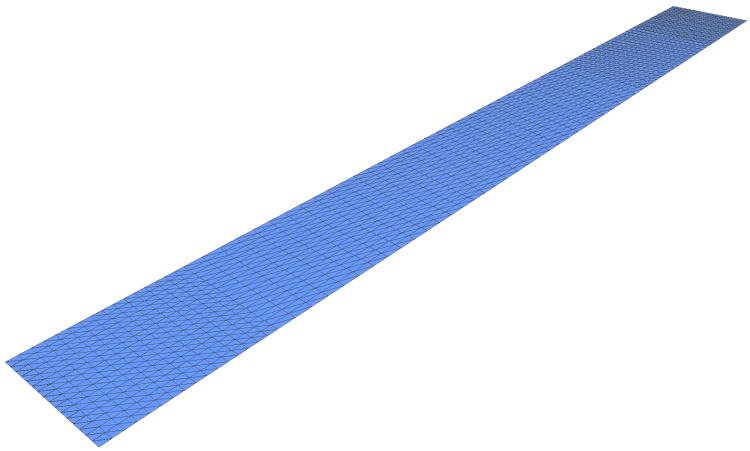

Figure 5.6: Wave profile at $t = 0.0$.

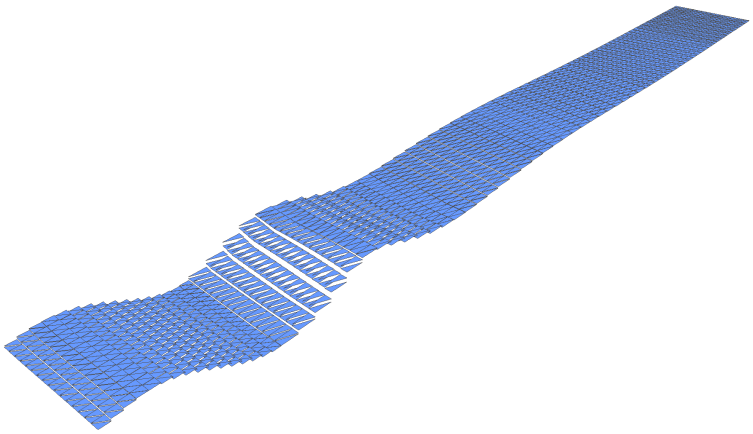

Figure 5.7: Wave profile at $t = 10.0$.

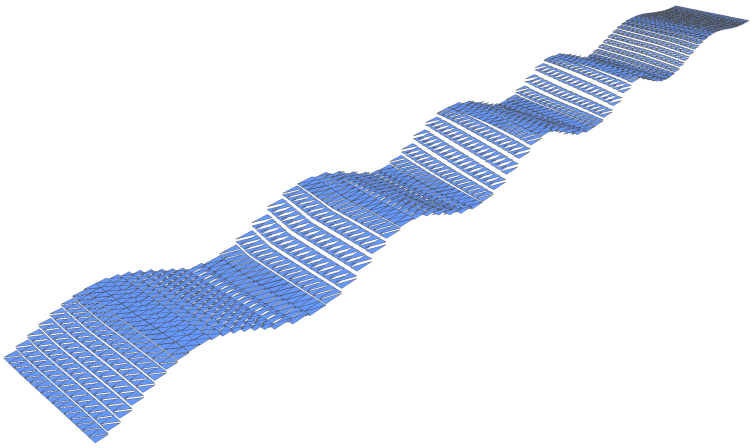

Figure 5.8: Wave profile at $t = 28.0$.

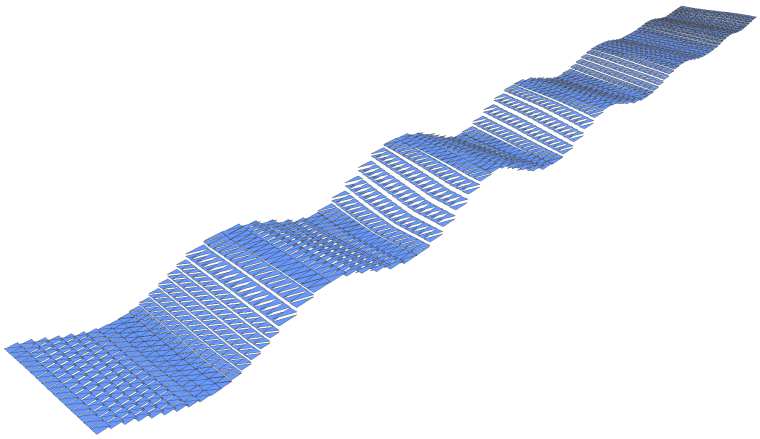

Figure 5.9: Wave profile at $t = 43.7$.

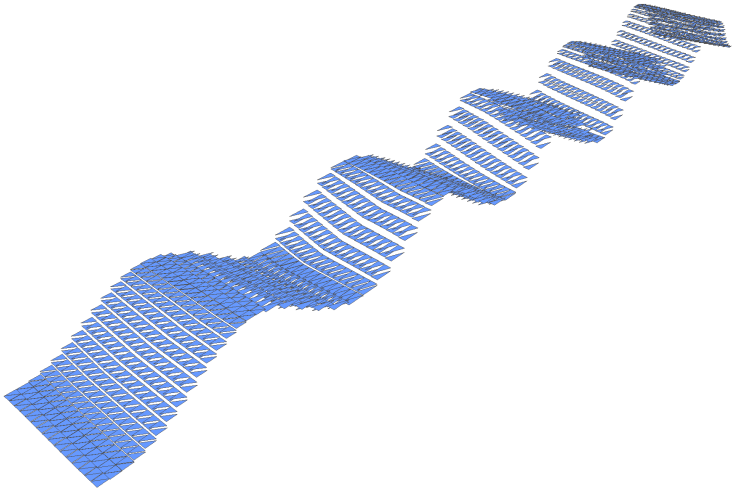

Figure 5.10: Wave profile at $t = 60.0$.

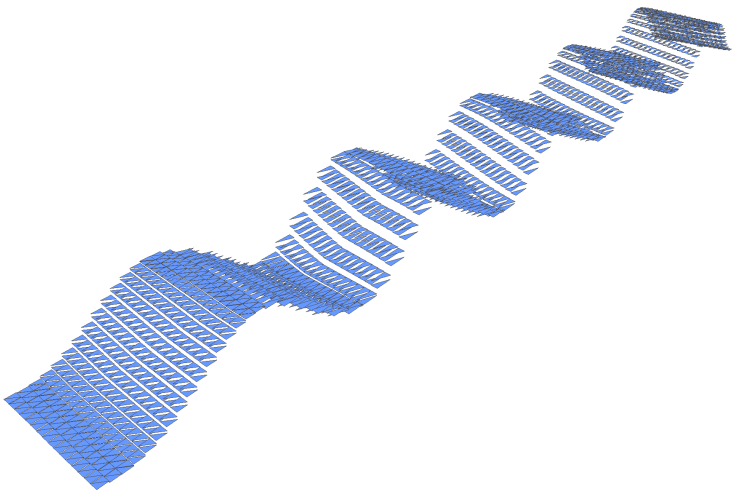

Figure 5.11: Wave profile at $t = 67.0$.

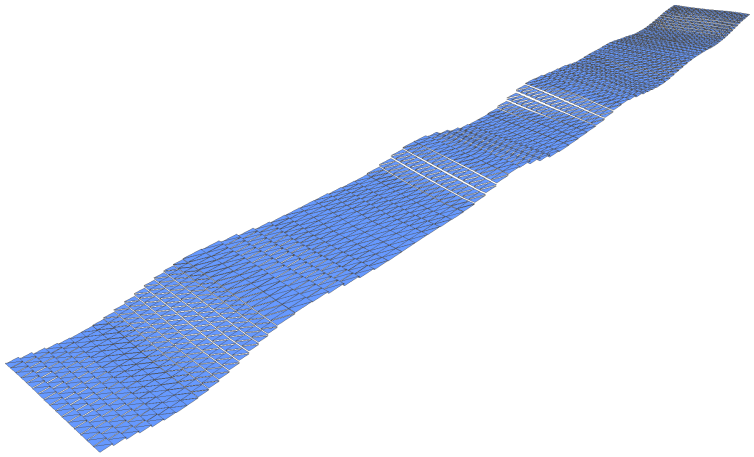

Figure 5.12: Wave profile at $t = 68.0$.

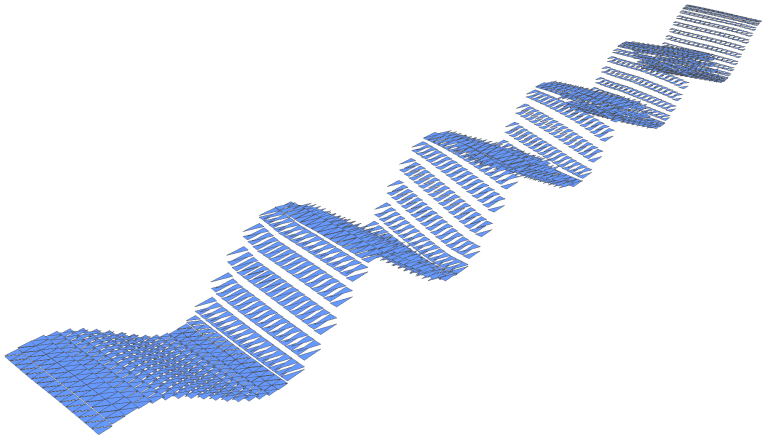

Figure 5.13: Wave profile at $t = 68.8$.

Chapter 6

Conclusion and outlook

We have presented a Galerkin Boundary Element Method for the solution of the time-dependent, linearized free surface flow problem in three dimensions. Data-sparse matrix approximation techniques were used for accelerating the solution of the boundary value problem. The results in Chapter 5 indicate that an improvement in runtime and memory complexity over a standard optimally preconditioned FEM is achievable.

There are several directions in which the work presented herein could be expanded. Among these are

- comparison of the obtained results to those of previously implemented methods and/or to real-world measurements;
- a full derivation of the underlying theory and a complete error analysis;
- further improvements to the data-sparse approximation algorithm and implementation in order to obtain a more clear-cut advantage over the FEM;
- generalization to the nonlinear case, where both the linearization of the boundary conditions and the approximation of the time-dependent domain $\Omega(t)$ by the stationary domain Ω may be undone;
- the incorporation of floating bodies into the mathematical model and thus a move towards the Numerical Wave Tank briefly described in the introduction.

Bibliography

[1] M. Bebendorf. AHMED. Another software library on hierarchical matrices for elliptic differential equations, Universität Leipzig, Fakultät für Mathematik und Informatik.

[2] M. Bebendorf. *Hierarchical Matrices*. Springer-Verlag Berlin Heidelberg, 2008.

[3] S. Börm and L. Grasedyck. HLib. A program library for hierarchical and H^2-matrices, Max Planck Institute for Mathematics in the Sciences, Leipzig.

[4] S. Börm, L. Grasedyck, and W. Hackbusch. Hierarchical matrices. Lecture Note 21, Max-Planck-Institut für Mathematik in den Naturwissenschaften Leipzig, 2003. Revised 2006.

[5] X. Cai, H. P. Langtangen, B. F. Nielsen, and A. Tveito. A finite element method for fully nonlinear water waves. *J. Comput. Physics*, 143(2):544–568, 1998.

[6] H. Engl and J. Synka. Mathematische Methoden der Kontinuumsmechanik. Vorlesungsskriptum, Institut für Industriemathematik, JKU Linz, November 2003.

[7] S. A. Goreinov, E. E. Tyrtyshnikov, and N. L. Zamarashkin. A theory of pseudoskeleton approximations. *Linear Algebra and its Applications*, 261:1–21, August 1997.

[8] S. Rjasanow and O. Steinbach. *The Fast Solution of Boundary Integral Equations (Mathematical and Analytical Techniques with Applications to Engineering)*. Springer-Verlag New York, Inc., 2007.

[9] I. Robertson and S. Sherwin. Free-surface flow simulation using *hp*/spectral elements. *J. Comput. Phys.*, 155(1):26–53, 1999.

[10] O. Steinbach. OSTBEM. A boundary element software package, University of Stuttgart, 2000.

[11] O. Steinbach. *Numerische Näherungsverfahren für elliptische Randwertprobleme. Finite Elemente und Randelemente.* B. G. Teubner Verlag, 2003.

[12] S. K. Tomar and J. J. W. van der Vegt. A Runge-Kutta discontinuous Galerkin method for linear free-surface gravity waves using high order velocity recovery. RICAM Report 2006-07, Austrian Academy of Sciences, RICAM Linz, February 2006.

[13] J.-H. Westhuis. *The Numerical Simulation of Nonlinear Waves in a Hydrodynamic Model Test Basin.* PhD thesis, Universiteit Twente, 2001.

[14] J. Whitham. *Linear and Nonlinear Waves.* John Wiley, New York, 1974.

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Diplom- bzw. Masterarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Haag, am 20. September 2008