The H^2-Wavelet Method

Daniel Alm, Helmut Harbrecht, Ulf Krämer

Institute of Mathematics
University of Basel
Rheinsprung 21
CH - 4051 Basel
Switzerland

Preprint No. 2013-17
June, 2013

www.math.unibas.ch

The $\mathcal{H}^2$-wavelet method

Daniel Alm[1]

Institut für Numerische Simulation, Universität Bonn, Wegelerstr. 6, 53115 Bonn, Germany

Helmut Harbrecht[2]

Mathematisches Institut, Universität Basel, Rheinsprung 21, 4051 Basel, Switzerland

Ulf Krämer[3]

Research & Development, CST AG, Bad Nauheimer Str. 19, 64289 Darmstadt, Germany

Abstract

In the present paper, we introduce the $\mathcal{H}^2$-wavelet method for the fast solution of nonlocal operator equations on unstructured meshes. On the given mesh, we construct a wavelet basis which provides vanishing moments with respect to the traces of polynomials in the space. With this basis at hand, the system matrix in wavelet coordinates is compressed to $\mathcal{O}(N \log N)$ relevant matrix coefficients, where N denotes the number of boundary elements. The compressed system matrix is computed with nearly linear complexity by using the fast $\mathcal{H}^2$-matrix approach. Numerical results in three spatial dimensions validate that we succeeded in developing a fast wavelet Galerkin scheme on unstructured triangular or quadrangular meshes.

Keywords: boundary element method, unstructured mesh, wavelet matrix compression

1. Introduction

Plenty of problems from physics can be modeled by means of partial differential equations. As only few of those problems can be solved analytically, computational means have to be employed. Notable examples are the *finite element method* (FEM) and *finite difference method*, which are based on a discretization of the entire simulation domain. Another approach is to transfer the partial differential equations to equivalent *boundary integral equations* which only need to be solved on the boundary of the original domain. This has the advantage of reducing the method's inherent dimensional complexity (as the boundary has one dimension less than the domain it encloses) as well as other desirable properties (like the possibility to obtain solutions on the – possibly unrestricted – *exterior* of the boundary).

The *boundary element method* (BEM) has emerged as a popular framework to solve boundary integral equations. Its concept is similar to that of the finite element method: the integral equation is reformulated as a variational problem, which is then solved for finite-dimensional subspaces of the original ansatz space. One notable drawback of the classic BEM, however, is that the underlying integral operator is in general non-local, which results in fully populated system matrices. Consequently, the required computation time and memory scale quadratically with the number of degrees of freedom.

Several methods have been developed to avoid this cost by providing an approximation of the matrix-vector multiplications necessary to solve the discretized problem. Examples are multipole and clustering methods [9, 13] and the closely related $\mathcal{H}$- and $\mathcal{H}^2$-matrix methods [3, 11] as well as the adaptive cross

[1]alm@ins.uni-bonn.de
[2]helmut.harbrecht@unibas.ch
[3]ulf.kraemer@cst.com

approximation [1]. Another class of methods are *wavelet methods* [2, 7, 23, 27] which are based on expressing the system matrix in terms of a multi-scale basis where it can be compressed to a sparse matrix without loss of accuracy. Unfortunately, most wavelet methods require a piecewise smooth parametrization of the boundary and are thus not applicable to arbitrary unstructured meshes. The so-called Tausch-White wavelets [30] are not affected by this problem, but still require the single-scale system matrix to be set up first.

The present paper introduces the $\mathcal{H}^2$*-wavelet method*. This method makes it possible to avoid assembling the single-scale matrix by employing $\mathcal{H}^2$-matrices to directly set up a quasi-sparse version of the system matrix in the Tausch-White wavelet basis. It can be adapted to a variety of problems and uses significantly less memory than the regular $\mathcal{H}^2$-matrix method. In addition, providing a quasi-sparse version of the system matrix (rather than just an approximation of the matrix-vector product) results in significantly faster matrix-vector multiplications and an easy means to employ preconditioning, which speeds up the solving process even further.

We mention that the theoretical foundation of the $\mathcal{H}^2$-wavelet method has been developed in [14, 20]. In this paper, we present its algorithmical realization with an emphasis on boundary integral equation in three spatial dimensions. In order to keep the scope of this paper narrow, its focus lies on solving the three-dimensional Laplace equation via the single-layer potential operator. This will allow us to make a few simplifications (mostly due to the symmetry of the operator and correspondingly the system matrix) in the underlying theory which helps to present the method in a straightforward way. The method can nevertheless be extended to other boundary integral operators and the key insights presented here are applicable to the method in general. In particular, more emphasis was spent on investigating the effectiveness of the preconditioner as well as finding optimal parameters for the algorithm. In addition, the algorithm was studied on an actual unstructured mesh for which no parameterization is available.

We will begin by introducing the boundary element method and the corresponding traditional Galerkin scheme in the next section. Tausch-White wavelets will be exposed in Section 3, followed by the $\mathcal{H}^2$-matrix method in Section 4. We will combine both in Section 5 and finally study the resulting algorithm's behavior in Section 6.

In order to reduce the use of insignificant constant factors throughout this paper, $C \lesssim D$ shall mean that C can be bounded by a multiple of D, independently of any parameters for C and D. $C \gtrsim D$ is equivalent to $D \lesssim C$ and $C \sim D$ is defined as $C \lesssim D$ and $C \gtrsim D$.

2. The Boundary Element Method

This section is dedicated to a short overview of the elementary components of the boundary element method. It also contains an explanation of the traditional Galerkin scheme usually employed to solve boundary integral equations.

2.1. The geometry

We begin by posing a few conditions on the boundary $\Gamma \subset \mathbb{R}^{n+1}$. It should be a closed, Lipschitz-continuous surface. We assume that a discretized representation $\Gamma_N = \bigcup_{i=1}^{N} \pi_i$ is available, consisting of n-dimensional convex simplices π_i, which we require to be either triangles or quadrangles. The intersection of two simplices should be empty or a lower-dimensional surface (e.g. a point or an edge). Points in the interior of quadrangles are obtained by bilinear interpolation. Therefore if a quadrangle's corners do not all lie in the same plane, the resulting simplices will be curved.

We also require *quasi-uniformity* of the discretization, i.e. the radii of the inscribed circles of all simplices should scale proportional to the step width h_N, which itself scales like $N^{-\frac{1}{n}}$. In addition, all simplices shall be oriented such that their normals are pointing outward.

2.2. The boundary integral operator

This paper is concerned with boundary integral equations of the form

$$A\rho = f \quad \text{on } \Gamma, \tag{2.1}$$

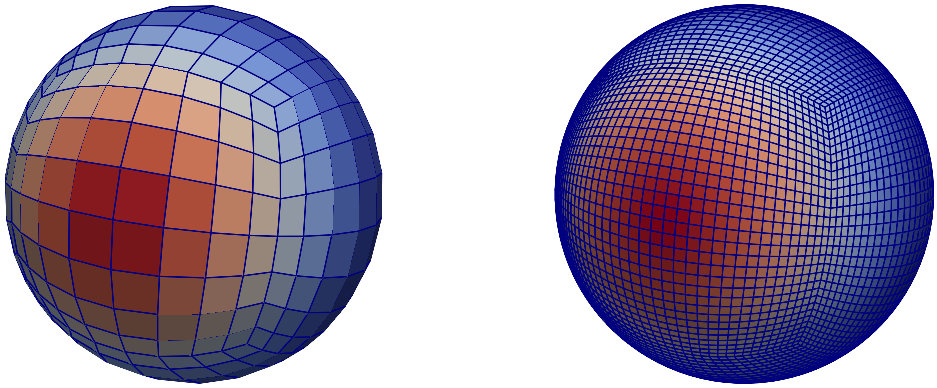

Figure 2.1: Quadrangulation of the sphere with three refinements (left) and five refinements (right).

where the boundary integral operator A can be written as

$$(A\rho)(x) := \int_\Gamma k(x,y)\rho(y)\mathrm{d}\sigma_y.$$

It should be a continuous and strictly coercive operator of order $2q$, i.e. map from $H^q(\Gamma)$ to $H^{-q}(\Gamma)$ and fulfill

$$\|\rho\|^2_{H^q(\Gamma)} \lesssim \langle A\rho,\rho\rangle_{L^2(\Gamma)}, \quad \langle A\rho, v\rangle_{L^2(\Gamma)} \lesssim \|\rho\|_{H^q(\Gamma)}\|v\|_{H^q(\Gamma)}$$

for all $\rho, v \in H^q(\Gamma)$. In addition, the kernel function should satisfy the decay condition

$$\left|\frac{\partial_x^\alpha \partial_y^\beta k(x,y)}{(|\alpha|+|\beta|)!}\right| \lesssim (s\|x-y\|)^{-(n+2q+|\alpha|+|\beta|)}, \quad s>0 \tag{2.2}$$

uniformly in $\alpha,\beta \in \mathbb{N}^{n+1}$. In particular, $k(x,y)$ is analytic in x and y except for a possible singularity at $x=y$. Note that we shall assume $q < \frac{1}{2}$ throughout the paper since we will restrict ourselves to a piecewise constant discretization.

One of the simplest and most popular examples for a partial differential equation that can be transferred to a boundary integral equation is the following: For a given $f \in H^{\frac{1}{2}}(\Gamma)$, find a function $u \in H^1(\Omega)$ which fulfills

$$\Delta u = 0 \text{ in } \Omega, \quad u = f \text{ on } \Gamma. \tag{2.3}$$

We introduce the *single layer potential operator* by

$$(\mathcal{V}\rho)(x) := \int_\Gamma k_S(x,y)\rho(y)\mathrm{d}\sigma_y, \quad x \in \Gamma$$

with $k_S(x,y) := \frac{1}{4\pi\|x-y\|}$ for $n=2$ and $k_S(x,y) := -\frac{1}{2\pi}\ln\|x-y\|$ for $n=1$. According to [25, 28], $\mathcal{V}$ is a symmetric and positive definite operator of order -1 that continuously maps from $H^{-\frac{1}{2}}(\Gamma)$ to $H^{\frac{1}{2}}(\Gamma)$ provided that Γ is Lipschitz continuous (and for $n=1$: $\mathrm{diam}(\Omega)<1$). Observing that the related *single layer potential*

$$\Phi_V(x) := \int_\Gamma k_S(x,y)\rho(y)\mathrm{d}\sigma_y, \quad x \in \Omega,$$

always satisfies $\Delta\Phi_V(x) = 0$, one can make the ansatz $u(x) = \Phi_V(x)$ to solve (2.3). By taking the trace, it is seen that the density function ρ is the solution of the Fredholm integral equation of the first kind

$$\mathcal{V}\rho = f \quad \text{on } \Gamma. \tag{2.4}$$

This is called the *indirect approach* as the solution function u is determined in two steps: first, the density function is determined, which is then used to evaluate the single layer potential. In the rest of this paper, we will mainly focus on the fast solution of (2.4). There are, of course, plenty of other boundary integral operators available, but treating them as well would exceed the scope of this paper.

2.3. The Galerkin method

In order to obtain a means of solving the integral equation (2.1), we employ the *Galerkin method*. It is based on the corresponding variational formulation:

$$\text{Find } \rho \in H^q(\Gamma) \text{ such that } \langle A\rho, v\rangle_{L^2(\Gamma)} = \langle f, v_N\rangle_{L^2(\Gamma)} \text{ for all } v \in H^q(\Gamma).$$

For a suitable finite-dimensional subspace of $V_N \subset H^q(\Gamma)$, one obtains a discrete variant of the problem:

$$\text{Find } \rho_N \in V_N \text{ such that } \langle A\rho_N, v_N\rangle_{L^2(\Gamma)} = \langle f, v_N\rangle_{L^2(\Gamma)} \text{ for all } v_N \in V_N. \tag{2.5}$$

We will use piecewise constant ansatz functions to constitute the space

$$V_N := \{v_N : \Gamma_N \to \mathbb{R} \mid \forall i : v_N|_{\pi_i} = \text{const.}\}.$$

In view of Céa's Lemma, [25] provides then the following error estimate for these spaces:

Proposition 2.1. *For the exact solution $\rho \in H^t(\Gamma)$ of (2.1) and the corresponding solution $\rho_N \in V_N$ (with the related step width h_N),*

$$\|\rho - \rho_N\|_{H^q(\Gamma)} \lesssim h_N^{t-q} \|\rho\|_{H^t(\Gamma)}$$

holds for $q \le t \le 1$. In particular, for $q = -\frac{1}{2}$ and $t = 1$, we have

$$\|\rho - \rho_N\|_{H^{-\frac{1}{2}}(\Gamma)} \lesssim h_N^{\frac{3}{2}} \|\rho\|_{H^1(\Gamma)}.$$

Note that by employing the Aubin-Nitsche trick (see e.g. [24, 25]), we can double the rate of convergence by measuring the error in $H^{-2}(\Gamma)$. Therefore, for the approximation $\Phi_{V,N}(x)$ of the single layer potential $\Phi_V(x)$ at a particular point $x \in \Omega$, we finally obtain the estimate

$$\begin{aligned} |\Phi_V(x) - \Phi_{V,N}(x)| &= \left| \int_\Gamma k_S(x,y)\big(\rho(y) - \rho_N(y)\big) \mathrm{d}\sigma_y \right| \\ &\le \|k_S(x,\cdot)\|_{H^2(\Gamma)} \, \|\rho - \rho_N\|_{H^{-2}(\Gamma)} \\ &\lesssim h_N^3 \, \|k_S(x,\cdot)\|_{H^2(\Gamma)} \, \|\rho\|_{H^1(\Gamma)}. \end{aligned}$$

It remains to present a suitable basis for the space V_N. We choose the L^2-normalized single-scale basis, i.e. each basis function $\phi_i, 1 \le i \le N$, is supported only on a single simplex:

$$\phi_i(x) := \begin{cases} \frac{1}{\sqrt{|\pi_i|}}, & x \in \pi_i, \\ 0, & \text{otherwise.} \end{cases}$$

Here, $|\pi_i|$ is defined as the area of π_i, leading to the desired L^2-orthonormality. Making the ansatz $\rho_N = \sum_{j=1}^N \rho_j^N \phi_j$ and testing with the basis in equation (2.5), we arrive at a system of linear equations

$$A^\phi \rho^\phi = f^\phi \tag{2.6}$$

where

$$A^\phi := [\langle A\phi_j, \phi_i\rangle]_{i,j=1}^N, \quad \rho^\phi := [\rho_i^N]_{i=1}^N, \quad f^\phi := [\langle f, \phi_i\rangle]_{i=1}^N.$$

A^ϕ is called *system matrix in the single-scale basis*. Setting up the system matrix (but not necessarily in this basis) or approximating the matrix-vector product $A^\phi \rho^\phi$ is one of the most involved aspects of the boundary element method and will concern us in the following sections.

2.4. Determining the entries of the system matrix

With the previous considerations, solving the discretized version of the boundary integral equation is reduced to setting up and solving the system of linear equations (2.6). The second part can be performed easily by means of iterative solvers, e.g. in the case of symmetric positive-definite matrices (as with the single-layer potential operator's matrix) the conjugate-gradient (CG) method.

The assembly of A^ϕ is significantly more complicated. Its entries are of the form

$$\langle A\phi_j, \phi_i \rangle = \int_\Gamma \int_\Gamma \phi_j(y) k(x,y) \phi_i(x) \mathrm{d}\sigma_x \mathrm{d}\sigma_y.$$

The kernel function does not vanish for distant panels, making the integral operator non-local. Consequently, A^ϕ will be fully populated, resulting in $\mathcal{O}(N^2)$ nonzero entries, which also provides a lower bound for the necessary computation time and memory requirements. In addition, these entries cannot be computed analytically in many cases, necessitating the use of quadrature formulae. The quadrature order needs to be adapted to the distance of the simplices in order to accommodate for the singularity at $x = y$. In the case of adjacent or identical panels, classic quadrature formulae cannot be used due to the singularity being inside the integration domain. This can be avoided by employing the Duffy trick. For all details concerning the numerical quadrature, we refer to [25].

3. Tausch-White Wavelets

In contrast to traditional wavelet constructions on surfaces (see [16] and the references therein) we cannot use a refinement strategy since the representation of the geometry automatically limits the finest level of any finite consideration to a single simplex. Hence, we will follow [30] and employ a coarsening procedure to define a multi-scale hierarchy

$$V_0 \subset V_1 \subset \cdots \subset V_{J-1} \subset V_J \equiv V_N. \tag{3.7}$$

3.1. The cluster tree

The multi-scale hierarchy will be linked to the discretization of the boundary by a *cluster tree*. Here, a *cluster* ν denotes the non-empty union $\nu = \bigcup_{i \in \mathbb{I}_\nu} \pi_i$ of a set of simplices π_i. The number of simplices contained by a cluster ν is written as $\#\nu$ and called its *cardinality*.

Definition 3.1 (Cluster tree)**.** *For a finite set T of clusters, a cluster ν is called* father cluster *of ν' (written $\nu' \prec \nu$), if $\nu' \subsetneq \nu$ holds and there are no other clusters $\nu'' \in T$ with $\nu' \subsetneq \nu'' \subsetneq \nu$. ν' is then called the* son *of ν. If a cluster has no sons, it is called a* leaf cluster*. The set of all leaf clusters is denoted by $L(T)$.*

T combined with the hierarchical ordering imposed by $\prec$ is called a cluster tree *of Γ_N, if the following additional conditions hold:*

1. *$\Gamma_N \in T$ is the only cluster in T without a father cluster (called the* root *$\hat{\nu}$ of T).*
2. *The intersection of two clusters with the same father cluster is either empty or a lower-dimensional set.*
3. *Each non-leaf cluster ν is identical to the union of its son clusters, i.e. $\nu = \bigcup_{\nu' \prec \nu} \nu'$.*

The level *of a cluster is the number j, such that clusters $\{\nu_i\}_{i=0}^{j-1}$ with $\nu \prec \nu_{j-1} \prec \cdots \prec \nu_0 = \Gamma_N$ exist. In particular, Γ_N is the only cluster with level 0 and there is a maximum level J, called the cluster tree's* depth*. For each level j, T_j denotes the set of all clusters belonging to that level.*

Remark 3.2. *The index sets $\mathbb{I}_\nu$ for non-leaf clusters ν will be ordered such that all indices from the index set $\mathbb{I}_{\nu'}$ of a corresponding son cluster ν' are adjacent in $\mathbb{I}_\nu$.*

Definition 3.1 is sufficient for obtaining a hierarchy of clusters, but it does not provide estimates for the sizes and cardinalities of clusters as demanded above. For this, we have another

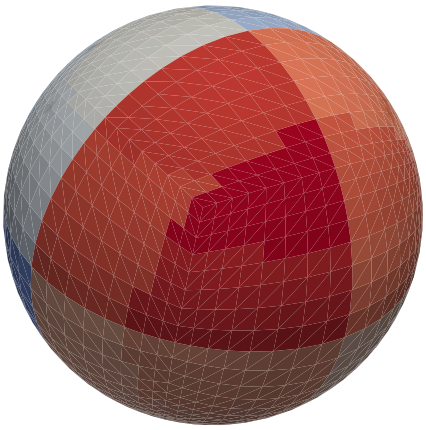

Figure 3.2: Example of a cardinality balanced clustering on the sphere. Different colors correspond to different clusters.

Definition 3.3 (Balanced 2^n-cluster tree)**.** *Let T be a cluster tree on $\Gamma_N \subset \mathbb{R}^{n+1}$ with depth J. It is called a* balanced 2^n-tree*, if its clusters ν all satisfy the following conditions (with j_ν being the level of ν):*

1. *ν has exactly 2^n sons (written $\ell_\nu = 2^n$) if $j_\nu < J$. It has no sons if $j_\nu = J$ (by definition of the depth),*
2. $\operatorname{diam}(\nu) \sim 2^{-\frac{j_\nu}{n}}$,
3. $\#\nu \sim 2^{(J-j_\nu)n}$.

There are several methods to create a cluster tree from a given boundary discretization. We will focus on a technique called *cardinality balanced clustering*. Here, the root cluster is split into two son clusters of identical (or similar) cardinality. This process is repeated recursively for the resulting son clusters until their cardinality falls below a certain threshold $\#_{\text{leaf,max}}$, called the *leaf size*. It remains to explain *how* a cluster's simplices are split up between the new son clusters. This is performed geometrically by using constructs from the following definition:

Definition 3.4 (Bounding box)**.** *Let ν be a cluster. Its* bounding box *B_ν is defined as the smallest axis-parallel cuboid that completely contains all its simplices.*

For the subdivision, the bounding box for the cluster's simplex midpoints simply is split along its longest edge such that the resulting two boxes both contain the same number of simplex midpoints. As the cluster cardinality halves thus with each level, we will obtain $\mathcal{O}(\log(N))$ levels in total. On each level, every simplex is considered a constant number of times for determining the splitting plane, resulting in a total runtime of $\mathcal{O}(N \log(N))$ for the clustering process. Examples for cluster trees created with this approach can be seen in Figure 3.2.

We finally transform this binary cluster tree into a balanced 2^n-tree by removing all clusters whose level is not a multiple of k. The resulting tree satisfies only the first and third property of a balanced 2^n-tree. But for quasi-uniform discretizations, the condition on the cluster diameters will be guaranteed asymptotically. This is sufficient for our computations. For sake of simplicity, for all subsequent proofs we will however assume the particular cluster trees to be balanced.

3.2. Constructing the wavelet basis

Now that we have created a balanced cluster tree, we can move on to construct a wavelet basis on the resulting hierarchical structure. To ensure that the diameters of the wavelet function's supports halves with each level, the support of a wavelet function on a specific level shall be restricted to a cluster of the same level. In addition, as the key ingredient for the matrix compression, we ask for the resulting wavelet functions to have *vanishing moments*.

Definition 3.5 (Vanishing moments)**.** *A function $\psi(x)$ has* vanishing moments up to order $\tilde{d}$*, if its moments*

$$\langle \psi, x^\alpha \rangle_{L^2(\Gamma)} = \int_\Gamma \psi(x) x^\alpha \mathrm{d}\sigma_x$$

are zero for all $\alpha \in \mathbb{N}_0^{n+1}$ with $|\alpha| < \tilde{d}$. We call $m_{\tilde{d}} := \#\{\alpha \in \mathbb{N}_0^{n+1} : |\alpha| < \tilde{d}\}$ the number of vanishing moments of ψ*.*

We begin by introducing a *two-scale* transform between basis functions on a cluster ν of level j. For this, we create *scaling functions* $\mathbf{\Phi}_j^\nu = \{\varphi_{j,k}^\nu\}$ and *wavelets* $\mathbf{\Psi}_j^\nu = \{\psi_{j,k}^\nu\}$ as linear combinations of the scaling functions $\mathbf{\Phi}_{j+1}^\nu$ of ν's son clusters. This results in the *refinement relation*

$$[\mathbf{\Phi}_j^\nu, \mathbf{\Psi}_j^\nu] := \mathbf{\Phi}_{j+1}^\nu [Q_{j,\Phi}^\nu, Q_{j,\Psi}^\nu]. \tag{3.8}$$

Remark 3.6. *We will frequently use the* sets *$\mathbf{\Phi}_j^\nu$ and $\mathbf{\Psi}_j^\nu$ as* row vectors *by indexing them in a particular order (see below).*

In order to provide vanishing moments and an orthonormal basis, we have to choose the transformation matrix carefully. A sensible choice is to obtain it from the QR-decomposition

$$\left(M_{j+1}^\nu\right)^T = QR =: [Q_{j,\Phi}^\nu, Q_{j,\Psi}^\nu] R \tag{3.9}$$

of the moment matrix

$$M_{j+1}^\nu := \langle x^\alpha, \mathbf{\Phi}_{j+1}^\nu \rangle_{|\alpha|<\tilde{d}} = \left[\int_\Gamma x^\alpha \varphi_{j+1,k}^\nu(x) \mathrm{d}\sigma_x \right]_{|\alpha|<\tilde{d},k}$$

of the son cluster's scaling functions. The moment matrix for the cluster's own scaling functions and wavelets is then

$$[M_j^{\nu,\Phi}, M_j^{\nu,\Psi}] = \langle x^\alpha, [\mathbf{\Phi}_j^\nu, \mathbf{\Psi}_j^\nu] \rangle_{|\alpha|<\tilde{d}} = \langle x^\alpha, \mathbf{\Phi}_{j+1}^\nu [Q_{j,\Phi}^\nu, Q_{j,\Psi}^\nu] \rangle_{|\alpha|<\tilde{d}} = M_{j+1}^\nu [Q_{j,\Phi}^\nu, Q_{j,\Psi}^\nu] = R^T. \tag{3.10}$$

As R^T is a lower left triangular matrix, the first $(k-1)$ entries in its k-th column are zero. This corresponds to $(k-1)$ vanishing moments for the k-th function generated by the transformation matrix $[Q_{j,\Phi}^\nu, Q_{j,\Psi}^\nu]$. By defining the first $m_{\tilde{d}}$ functions as scaling functions and the remaining as wavelets, we obtain wavelets with vanishing moments up to at least order $\tilde{d}$. In addition, the QR-decomposition also causes some scaling functions to have a few vanishing moments.

Remark 3.7. *Each cluster has an at most constant number C_Q of scaling functions and wavelets: For a particular cluster ν, their number is identical to the cardinality of $\mathbf{\Phi}_{j+1}^\nu$. For leaf clusters, this number is bounded by the leaf size $\#_{\mathrm{leaf,max}}$. For non-leaf clusters, it is bounded by the number of scaling functions provided from all its son clusters. As there are at most 2^n son clusters with a maximum of $m_{\tilde{d}}$ scaling functions each, we obtain a bound of $2^n m_{\tilde{d}}$ for non-leaf clusters. Note that, if $\mathbf{\Phi}_{j+1}^\nu$ has at most $m_{\tilde{d}}$ elements, a cluster will not provide any wavelets at all – all functions it provides will be scaling functions.*

For leaf clusters, we take the single-scale basis functions $\mathbf{\Phi}_J^\nu := \{\phi_i : i \in \mathbb{I}_\nu\}$ to make up for the lack of son clusters that could provide scaling functions. The scaling functions of all clusters on a specific level j then make up the function spaces

$$V_j := \mathrm{span}\{\varphi_{j,k}^\nu : \nu \in T_j\}, \tag{3.11}$$

while the wavelets are used to create the detail spaces

$$W_j := \mathrm{span}\{\psi_{j,k}^\nu : \nu \in T_j\} = V_{j+1} \overset{\perp}{\ominus} V_j. \tag{3.12}$$

Combining the scaling functions of the root cluster with all clusters' wavelets yields the final wavelet basis

$$\mathbf{\Psi}_N := \mathbf{\Phi}_0^{\Gamma_N} \cup \bigcup_{\nu \in T} \mathbf{\Psi}_{j_\nu}^\nu. \tag{3.13}$$

Writing $\mathbf{\Psi}_N = \{\vartheta_k : 1 \le k \le N\} = \{\vartheta_k : k \in \mathbb{I}_N^\Psi\}$, where ϑ_k is either a wavelet or a scaling function on the root cluster, we can establish a unique indexing of all the functions in the wavelet basis. In a few cases, this allows us to conveniently write ψ_k or $\psi_{j,k}$ rather than $\psi_{j,k}^\nu$. Analogously, we also collect all scaling functions in the set

$$\mathbf{\Phi}_N := \bigcup_{\nu \in T} \mathbf{\Phi}_{j_\nu}^\nu = \{\vartheta_k : k \in \mathbb{I}_N^\Phi\}.$$

Note that $\mathbf{\Phi}_N$ is *not* a basis of V_N – it contains scaling functions from all levels, although those on coarser levels are just linear combinations of those on finer levels. The indexing naturally induces an order on the function sets $\mathbf{\Phi}_N$ and $\mathbf{\Psi}_N$, which we choose to be *level-dependent:* Functions belonging to a particular cluster are grouped together, with those on finer levels having higher indices.

We collect the scaling function and wavelet indices for a particular cluster ν in the sets $\mathbb{I}_\nu^\Phi \subset \mathbb{I}_N^\Phi$ and $\mathbb{I}_\nu^\Psi \subset \mathbb{I}_N^\Psi$. This allows us to define vectors $f_\mu^\Psi = \left[\langle f, \psi_i \rangle_{L^2(\Gamma)}\right]_{i \in \mathbb{I}_\mu^\Psi}$ and matrix blocks

$$A_{\mu,\nu}^{\Phi,\Phi} = \left[\langle A\varphi_{i'}, \varphi_i \rangle_{L^2(\Gamma)}\right]_{i \in \mathbb{I}_\mu^\Phi, i' \in \mathbb{I}_\nu^\Phi}, \qquad A_{\mu,\nu}^{\Phi,\Psi} = \left[\langle A\varphi_{i'}, \psi_i \rangle_{L^2(\Gamma)}\right]_{i \in \mathbb{I}_\mu^\Phi, i' \in \mathbb{I}_\nu^\Psi}$$
$$A_{\mu,\nu}^{\Psi,\Phi} = \left[\langle A\psi_{i'}, \varphi_i \rangle_{L^2(\Gamma)}\right]_{i \in \mathbb{I}_\mu^\Psi, i' \in \mathbb{I}_\nu^\Phi}, \qquad A_{\mu,\nu}^{\Psi,\Psi} = \left[\langle A\psi_{i'}, \psi_i \rangle_{L^2(\Gamma)}\right]_{i \in \mathbb{I}_\mu^\Psi, i' \in \mathbb{I}_\nu^\Psi}$$

of coefficients in the multi-scale basis. For a cluster pair (μ, ν) we call μ the *row cluster*, because the indices in $\mathbb{I}_\mu$ then correspond to rows of the matrix. Analogously, ν is called the *column cluster*.

Remark 3.8. *In order to provide a uniform number of wavelets per cluster, each leaf cluster should contain approximately $2^n m_{\tilde{d}}$ simplices. This corresponds to setting $\#_{\text{leaf,max}} \sim 2^n m_{\tilde{d}}$, respectively.*

Therefore, each cluster's moment matrix will also have at most C_Q columns, which allows us to formulate the following

Lemma 3.9. *Constructing the wavelet basis on a balanced 2^n-cluster tree has storage and computational cost of $\mathcal{O}(N)$.*

Proof. For each leaf cluster, we compute the corresponding moment matrix (whose row and column counts are each bounded by a constant) in the single-scale basis, resulting in a constant effort per cluster. For non-leaf clusters, collecting the son cluster's moments (which have already been computed) also takes constant time. Computing the QR decomposition of the constant-size moment matrix again takes $\mathcal{O}(1)$ operations, as does multiplying it with $Q_{j,\tilde{\Phi}}^\nu$. Employing finally that we have a total number of $\mathcal{O}(N)$ clusters and a constant effort per cluster, we obtain a total runtime of $\mathcal{O}(N)$ operations. □

3.3. Properties of the wavelets

By construction, the Tausch-White wavelets satisfy the following properties (cf. [14, 30]).

Theorem 3.10. *The spaces V_j as defined in equation* (3.11) *fulfill the desired multi-scale hierarchy* (3.7)*, where the respective complement spaces W_j from* (3.12) *fulfill $V_{j+1} = V_j \overset{\perp}{\oplus} W_j$ for all $j = 0, 1, \ldots, J-1$. The associated wavelet basis $\mathbf{\Psi}_N$ defined in* (3.13) *constitutes an L^2-orthonormal basis of V_J. In particular:*

1. *The diameter of the support of a wavelet on level j behaves like 2^{-j}.*
2. *The number of all wavelets on level j behaves like 2^{jn}.*
3. *The wavelets have vanishing moments up to order $\tilde{d}$.*

Remark 3.11. *Due to $W_j \subset V_N$ and $V_0 \subset V_N$, we conclude that each basis function is a linear combinations of the single-scale basis functions on the finest level. Especially, the related coefficient vectors $\omega^{j,k}$ in*

$$\psi_{j,k} = \sum_{i=1}^N \omega_i^{j,k} \phi_i \quad \text{and} \quad \varphi_{0,k} = \sum_{i=1}^N \omega_i^{0,k} \phi_i \tag{3.14}$$

are pairwise orthonormal.

Later on, the following bound for the wavelet's L^1-norm will be essential:

Lemma 3.12. *A wavelet $\psi_{j,k}^{\nu}$ on the cluster ν fulfills*

$$\int_{\Gamma_N} |\psi_{j,k}^{\nu}(x)| \mathrm{d}\sigma_x \lesssim 2^{-j\frac{n}{2}}.$$

The same holds for scaling functions $\varphi_{j,k}^{\nu}$.

Proof. The Cauchy-Schwartz inequality implies

$$\left(\int_{\Gamma_N} |\psi_{j,k}^{\nu}(x)| \mathrm{d}\sigma_x \right)^2 \leq \|\psi_{j,k}^{\nu}\|_{L^2(\Gamma)}^2 \int_{\nu} 1 \mathrm{d}\sigma_x = \sum_{i \in \mathbb{I}_{j,k}} \int_{\pi_i} 1 \mathrm{d}\sigma_x.$$

For a quasi-uniform discretization we have $|\pi_i| \sim h_N^n$. For a 2^n-tree of depth J, we also have $h_N \sim 2^{-J}$, resulting in $|\pi_i| \sim 2^{-Jn}$. The cluster ν has cardinality of $\sim 2^{(J-j)n}$, which provides the bound $\mathbb{I}_{j,k} \lesssim 2^{(J-j)n}$. Combining these estimates finally yields

$$\left(\int_{\Gamma_N} |\psi_{j,k}^{\nu}(x)| \mathrm{d}\sigma_x \right)^2 \lesssim 2^{(J-j)n} 2^{-Jn} = 2^{-jn},$$

which is just the squared version of the claim. □

3.4. Transforming between bases

In order to numerically switch between the multi-scale basis and the single-scale basis, we need to the *discrete wavelet transform* and its inverse. Hence, given a representation $f = \sum_{i=1}^{N} f_i^{\phi} \phi_i$ in the single-scale basis, we are looking for the related representation in the $f = \sum_{k \in \mathbb{I}_N^{\Psi}} f_k^{\Psi} \psi_k$. For sake of a simpler notation, let $f^{\phi} := [f_i^{\phi}]_{i=1}^{N}$ and $f^{\Psi} := [f_k^{\Psi}]_{k \in \mathbb{I}_N^{\Psi}}$ denote the associated coefficient vectors.

Remark 3.13. *In order to represent f in the multi-scale basis, we also need the scaling functions $\varphi_{0,k}^{\Gamma_N}$ of the root cluster. For a more convenient notation, they are implicitly included when we write ψ_k, $k \in \mathbb{I}_N^{\Psi}$.*

We transform between bases by means of the *discrete wavelet transform* (DWT). It is based on recursively applying the refinement relation (3.8) to the inner products

$$\langle f, [\boldsymbol{\Phi}_j^{\nu}, \boldsymbol{\Psi}_j^{\nu}] \rangle_{L^2(\Gamma)} = \langle f, \boldsymbol{\Phi}_{j+1}^{\nu} [Q_{j,\Phi}^{\nu}, Q_{j,\Psi}^{\nu}] \rangle_{L^2(\Gamma)} = \langle f, \boldsymbol{\Phi}_{j+1}^{\nu} \rangle_{L^2(\Gamma)} [Q_{j,\Phi}^{\nu}, Q_{j,\Psi}^{\nu}]. \tag{3.15}$$

On the finest level, the elements $\langle f, \boldsymbol{\Phi}_{J+1}^{\nu} \rangle_{L^2(\Gamma)}$ correspond to entries of f^{ϕ}. Recursively applying equation (3.15) yields all the coefficients $\langle f, \boldsymbol{\Psi}_j^{\nu} \rangle_{L^2(\Gamma)}$ (plus $\langle f, \boldsymbol{\Phi}_0^{\Gamma_N} \rangle_{L^2(\Gamma)}$) required for a representation of f in the wavelet basis. This concept is formulated in Algorithm 3.1.

Algorithm 3.1: Discrete wavelet transform

Data: Single-scale coefficient vector f^{ϕ} of the function $f = \sum_{i=1}^{N} f_i^{\phi} \phi_i$, cluster tree T, and transformation matrices $Q_{j,\Phi}^{\nu}$, $Q_{j,\Psi}^{\nu}$.

Result: Multi-scale coefficient vector f^{Ψ} with $f = \sum_{k \in \mathbb{I}_N^{\Psi}} f_k^{\Psi} \psi_k$, stored as inner products $\langle (\boldsymbol{\Phi}_0^{\Gamma_N})^T, f \rangle_{L^2(\Gamma)}$ and $\langle (\boldsymbol{\Psi}_j^{\nu})^T, f \rangle_{L^2(\Gamma)}$.

begin

store $\langle (\boldsymbol{\Phi}_0^{\Gamma_N})^T, f \rangle_{L^2(\Gamma)} :=$ `transformForCluster`(Γ_N);

Function transformForCluster(ν)

begin

  if ν *is a leaf cluster with simplices* $\{\pi_{i_1}, \dots, \pi_{i_{\#\nu}}\}$ **then**

    read $f^{\nu}_{j_\nu+1} := \left[f^{\phi}_{i_k}\right]_{k=1}^{\#\nu}$;

  else

    for $\nu_{\text{son}} \prec \nu$ **do**

      execute `transformForCluster`(ν_{son}) and append the result to $f^{\nu}_{j_\nu+1}$;

  store $\langle (\mathbf{\Psi}^{\nu}_{j_\nu})^T, f\rangle_{L^2(\Gamma)} := (Q^{\nu}_{j_\nu,\Psi})^T f^{\nu}_{j_\nu+1}$;

  return $(Q^{\nu}_{j_\nu,\Phi})^T f^{\nu}_{j_\nu+1}$;

Remark 3.14. *Algorithm 3.1 uses a transposed version of* (3.15) *in order to keep the column vector structure of* f^{ϕ} *and* f^{Ψ}.

The inverse operation is executed by performing the DWT's steps in the opposite order and direction: On each cluster, we compute $\langle f, \mathbf{\Phi}^{\nu}_{j+1}\rangle_{L^2(\Gamma)} = \langle f, [\mathbf{\Phi}^{\nu}_{j}, \mathbf{\Psi}^{\nu}_{j}]\rangle_{L^2(\Gamma)}[Q^{\nu}_{j,\Phi}, Q^{\nu}_{j,\Psi}]^T$ to obtain either the son clusters' scaling function inner products or (for leaf clusters) coefficients of f^{ϕ}. The necessary coefficients $\langle (\mathbf{\Psi}^{\nu}_{j})^T, f\rangle_{L^2(\Gamma)}$ (as well as $\langle (\mathbf{\Phi}_0^{\Gamma_N})^T, f\rangle_{L^2(\Gamma)}$) are stored in f^{Ψ}, while the $\langle (\mathbf{\Phi}^{\nu}_{j})^T, f\rangle_{L^2(\Gamma)}$ for non-root clusters have been computed in the father cluster's recursion step. The corresponding algorithm is omitted for sake of brevity.

Lemma 3.15. *The transformation from the single-scale basis to the multi-scale basis or vice versa can be performed in linear time.*

Proof. Similarly to the proof of Lemma 3.9, Algorithm 3.1 performs two constant-size (and thus constant-time) matrix-vector multiplications on each of the $\mathcal{O}(N)$ clusters in total. The runtime estimate for the inverse transform is derived in complete analogy. □

3.5. Matrix compression

As the wavelets already provide vanishing moments, we can employ the decay condition (2.2) which leads to the estimate

$$\left|\langle A\psi_{j',k'}, \psi_{j,k}\rangle_{L^2(\Gamma)}\right| \lesssim \frac{2^{-(j+j')(\tilde{d}+\frac{n}{2})}}{\operatorname{dist}(\Theta_{j,k}, \Theta_{j',k'})^{n+2q+2\tilde{d}}}.$$

Due to this estimate, the system matrix $A^{\Psi} = \left[\langle A\psi_{j',k'}, \psi_{j,k}\rangle\right]_{k,k'\in\mathbb{I}_N^{\Psi}}$ becomes numerically sparse in wavelet coordinates. To indentify negligible entries *before actually computing them*, we formulate a criterion on the distance of wavelet supports. We will remove all entries for which the distance of the corresponding wavelet supports is larger than the *cut-off parameter*

$$\mathcal{B}_{j,j'} := a \max\left\{2^{-\min\{j,j'\}}, 2^{\frac{2J(d'-q)-(j+j')(d'+\tilde{d})}{2(\tilde{d}+q)}}\right\} \tag{3.16}$$

with $a > 1$ and $1 < d' < \tilde{d} + 2q$. The requirement $a > 1$ is necessary for proving some of the following properties, but can easily be avoided by re-scaling the entire mesh with a constant factor. Figure 3.3 shows a system matrix in the multi-scale basis for which this compression scheme has been applied.

Theorem 3.16 (Cut-off condition)**.** *Let the cut-off parameter* $\mathcal{B}_{j,j'}$ *given as in* (3.16)*. Then, setting all entries* $A^{\Psi}_{(j,k),(j',k')}$ *to zero for which the* cut-off condition $\operatorname{dist}(\Theta_{j,k}, \Theta_{j',k'}) > \mathcal{B}_{j,j'}$ *holds removes all but* $\mathcal{O}(N\log(N))$ *entries. This does not affect the stability and convergence order of the wavelet Galerkin scheme.*

Proof. For the stability and convergence proofs, we refer to [7, 27]. The other assertion is also proved there. However, we present it here since we will refer to it later on. To this end, note that most wavelet supports

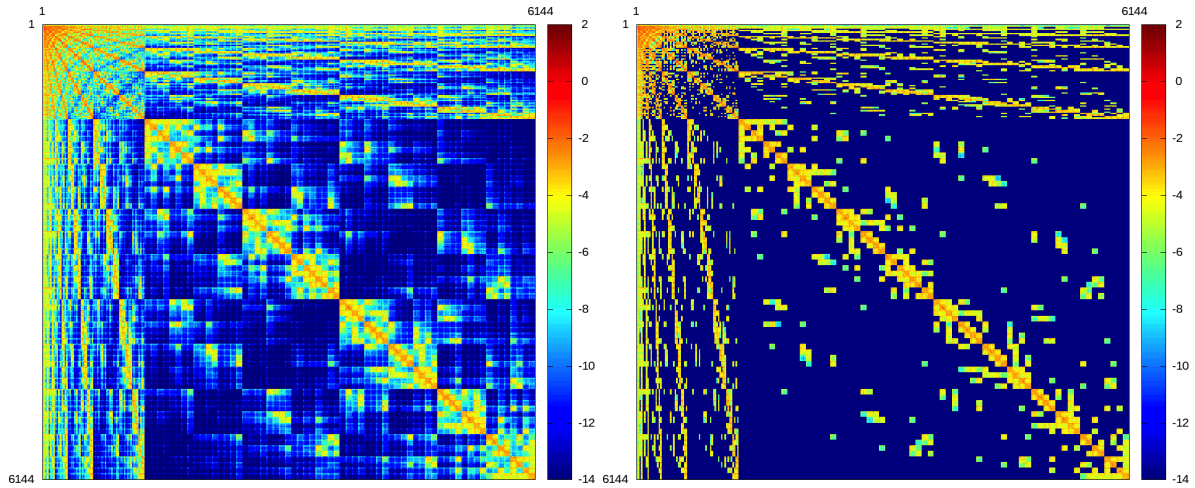

Figure 3.3: System matrix for the sphere (after five refinements, i.e. $N = 6144$) before (left) and after (right) applying the wavelet compression.

are identical to the corresponding cluster. Therefore, it is sufficient to count the cluster pairs closer than the cut-off parameter, as each cluster provides at most C_Q wavelets.

For a particular pair of levels j and j', there are $\sim 2^{(j+j')n}\mathcal{B}_{j,j'}^n$ cluster pairs: For a cluster ν of level $j' \leq j$, we can fit $\sim \left(\frac{\mathcal{B}_{j,j'}}{2^{-j}}\right)^n = 2^{jn}\mathcal{B}_{j,j'}^n$ clusters of level j (which each have diam $\sim 2^{-j}$) in a $\mathcal{B}_{j,j'}$-ball around ν. We obtain the desired estimate by multiplying with $2^{nj'}$, which is the total number of clusters of level j'.

With $M := \frac{d'+\tilde{d}}{2(\tilde{d}+q)}$, we can write

$$2^{\frac{2J(d'-q)-(j+j')(d'+\tilde{d})}{2(\tilde{d}+q)}} = 2^{-J}2^{(J-j)M}2^{(J-j')M}$$

to formulate the cut-off parameter as

$$\mathcal{B}_{j,j'} = a \max\left\{2^{-\min\{j,j'\}}, 2^{-J}2^{(J-j)M}2^{(J-j')M}\right\}.$$

In the case $2^{-\min\{j,j'\}} \leq 2^{-J}2^{(J-j)M}2^{(J-j')M}$, we thus have

$$\sum_{j,j'=0}^{J} 2^{(j+j')n} \max\left\{2^{-\min\{jn,j'n\}}, 2^{-Jn}2^{(J-j)Mn}2^{(J-j')Mn}\right\} \lesssim \sum_{j,j'=0}^{J} 2^{-Jn}2^{(J-j)(M-1)n}2^{(J-j')(M-1)n} \lesssim 2^{Jn},$$

thanks to $M < 1$ and thus $2^{(M-1)n} < 1$. In the other case, we get

$$\sum_{j,j'=0}^{J} 2^{(j+j')n} \max\left\{2^{-\min\{jn,jn'\}}, 2^{-Jn}2^{(J-j)Mn}2^{(J-j')Mn}\right\} \lesssim \sum_{j,j'=0}^{J} 2^{(j+j')n}2^{-\min\{jn,j'n\}} \lesssim J2^{Jn}.$$

With $2^{Jn} \sim N$ and $J \sim \log(N)$ we have in total $\mathcal{O}(N\log(N))$ cluster pairs closer than the corresponding cut-off parameter, with a constant number of entries for each pair. □

If $\operatorname{dist}(\mu,\nu) > \mathcal{B}_{j,j'}$ holds for a cluster pair (μ,ν) with corresponding levels j and j', the pair is said to *satisfy the cut-off condition*. Since, however, computing the distance between the convex hulls of wavelet supports (or clusters, which is equivalent) is not easy, we will therefore use the simplified criterion $\operatorname{dist}(B_\mu, B_\nu) >$

$\mathcal{B}_{j,j'}$ for wavelets $\psi^{\mu}_{j,k}$ and $\psi^{\nu}_{j',k'}$. Thanks to $\psi^{\mu}_{j,k} \subseteq \mu \subset B_\mu$, we have $\operatorname{dist}(\Theta_{j,k}, \Theta_{j',k'}) \geq \operatorname{dist}(B_\mu, B_\nu)$, resulting in more remaining entries, i.e. preserved accuracy but weaker compression. Asymptotically, both estimates will be identical, however, as comparing bounding boxes and convex hulls does not make a substantial difference.

Remark 3.17. *By the nature of this compression scheme, only complete sub-blocks of the system matrix in the wavelet basis are stored. It is therefore advisable to implement a block-sparse matrix data structure which does not store single entries (with the corresponding indices) but rather entire matrix blocks. This is significantly more efficient than a per-entry sparse matrix. On the other hand, this approach prohibits an* a-posteriori compression *[7, 17], where all matrix* entries *(as opposed to matrix* blocks*) below a certain level-dependent threshold are discarded, because it would destroy the block structure of the matrix. As the a-posteriori compression does not improve the asymptotic memory cost of $\mathcal{O}(N \log(N))$, anyway, it will not be employed in this paper. In addition, the overhead of having to use a per-entry sparse matrix implementation can easily offset the benefits of a slightly better compression ratio.*

3.6. Determining the sparsity pattern

For a given pair of clusters, we can now determine whether the corresponding entries need to be calculated. As there are $\mathcal{O}(N)$ clusters, naively checking the cut-off criterion for all pairs would still take $\mathcal{O}(N^2)$ operations, however. We therefore need a smarter means to enumerate the non-omittable cluster pairs. For this purpose, we first state the transferability of the cut-off condition to son clusters (cf. [7]):

Lemma 3.18. *Let μ and ν be clusters satisfying the cut-off criterion $\operatorname{dist}(B_\mu, B_\nu) > \mathcal{B}_{j_\mu, j_\nu}$. Then, for possible son clusters $\mu_{\text{son}} \prec \mu$ and $\nu_{\text{son}} \prec \nu$, we have $\operatorname{dist}(B_{\mu_{\text{son}}}, B_\nu) > \mathcal{B}_{j_{\mu_{\text{son}}}, j_\nu}$, $\operatorname{dist}(B_\mu, B_{\nu_{\text{son}}}) > \mathcal{B}_{j_\mu, j_{\nu_{\text{son}}}}$, and $\operatorname{dist}(B_{\mu_{\text{son}}}, B_{\nu_{\text{son}}}) > \mathcal{B}_{j_{\mu_{\text{son}}}, j_{\nu_{\text{son}}}}$.*

This lemma allows us to avoid inspecting cluster pairs whose father clusters already satisfy the cut-off condition. It leads to the following recursive scheme (see also Algorithm 3.2): For given clusters μ and ν, we check if (μ, ν) satisfies the cut-off condition. If not, we recursively perform the same check with (μ_{son}, ν) for each son μ_{son} of μ, until either the condition is fulfilled or we arrive at a leaf cluster. If we execute this procedure for all possible column clusters $\nu \in T$. The result is a list of all cluster pairs that do not satisfy the cut-off condition (i.e. are not negligible). The efficient computation of these non-negligible entries is then the topic of Section 5.

Algorithm 3.2: Recursive cut-off criterion check

```
Data: Cluster tree T, cut-off parameters a and d'.
Result: Sparse matrix Ã^Ψ with ones in all blocks where the cut-off condition is not satisfied.
begin
    Ã^Ψ := [0]_{k,k' ∈ I_N^Ψ};
    for ν ∈ T do
        checkCutOffCriterionRecursively(Γ_N, ν);
```

Function checkCutOffCriterionRecursively(μ, ν)

```
begin
    if dist(B_μ, B_ν) ≤ B_{j_μ, j_ν} then
        Ã^Ψ_{μ,ν} := [1]_{k ∈ I_μ^Ψ, k' ∈ I_ν^Ψ};
        if μ is not a leaf cluster then
            for μ_son ≺ μ do
                checkCutOffCriterionRecursively(μ_son, ν);
```

Lemma 3.19. *Algorithm 3.2 has runtime $\mathcal{O}(N \log(N))$.*

Proof. `checkCutOffCriterionRecursively` only gets called for each cluster pair closer than the cut-off parameter (of which there are $\mathcal{O}(N\log(N))$ – see the proof of Theorem 3.16) and for the immediate sons of the row clusters of such pairs. It does not get called for *all* cluster pairs. □

Remark 3.20. *In the case of symmetric system matrices (as with the single-layer potential operator), determining the sparsity pattern can be sped up by only computing its upper right part. To achieve this, the recursion is stopped when the levels of μ and ν are identical.*

3.7. Wavelet preconditioning

If the boundary integral operator A under consideration has an order $2q$ which is different from zero, then preconditioning becomes an issue for the iterative solver. Fortunatelty, wavelets offer a built-in preconditioning. The following result has been proven in [14].

Lemma 3.21. *Assume that $|q| < \frac{1}{2}$. Then, integral operator's system matrix in the multi-scale basis A^Ψ can be preconditioned by scaling with its diagonal:*

$$\operatorname{cond}\Big(\big(\operatorname{diag}(A^\Psi)\big)^{-\frac{1}{2}} A^\Psi \big(\operatorname{diag}(A^\Psi)\big)^{-\frac{1}{2}}\Big) \sim 1.$$

It holds $q = -\frac{1}{2}$ in case of the single layer operator. Here, the condition number of the preconditioned system matrix grows like $(\log(N))^2$, cf. [22].

4. The $\mathcal{H}^2$-Matrix Method

This section consists of a short, but complete – in the sense that it explains all steps necessary for approximating the matrix-vector product $A^\phi f^\phi$ – description of the $\mathcal{H}^2$*-matrix method*. The focus will be on the aspects that are useful for the $\mathcal{H}^2$-wavelet method explained in Section 5, however. In particular, we will assume the clusters corresponding to *rows* and *columns* of the system matrix to belong to the same tree (which is natural for symmetric matrices) and assume a *fixed* interpolation order for all clusters (as the $\mathcal{H}^2$-wavelet method does not benefit from variable interpolation orders).

The matrix-vector product is approximated by splitting the system matrix into two parts $A^\phi \approx A^\phi_{\text{near}} + A^\phi_{\text{far}}$. The nearfield A^ϕ_{near} will be computed classically. It can be stored (and applied) efficiently via a block-sparse matrix. The farfield matrix A^ϕ_{far} is subdivided into several blocks corresponding to cluster pairs that satisfy a certain *admissibility condition*. Each of these blocks will not be stored in full, but approximated as a product of lower-rank matrices. This approach is based on the *degenerated kernel expansion*, in which the kernel function $k(x, y)$ is separated into a product of two functions solely depending on x and y, respectively. The kernel expansion can be realized, for example, via spherical harmonics (as with the *fast multipole method*) or Taylor polynomials. In this paper, we will resort to the more computationally intensive but also more flexible means of polynomial interpolation.

4.1. Approximating the kernel function

A crucial aspect of the $\mathcal{H}^2$-method is the low-rank approximation of the kernel function for distant evaluation points. Originally, a Taylor expansion of the kernel function was employed to accomplish this task. In this paper, we will focus on Chebyshev interpolation in the bounding boxes B_μ and B_ν of admissible cluster pairs (see below).

To ensure optimal stability for interpolating on the interval $[-1, 1]$ (see [25, 29]), we choose the roots

$$x_s = \cos\left(\pi \frac{2s+1}{2p+2}\right), \quad s = 0, \ldots, p$$

of the p-th order Chebyshev polynomial as interpolation nodes and define the associated Lagrange polynomials of order p

$$\mathcal{L}_s(x) := \prod_{t=0, t\neq s}^{p} \frac{x - x_t}{x_s - x_t}.$$

These polynomials can be used for interpolating on an arbitrary interval $[a,b]$ by employing an affine transform

$$\left(T_{\text{aff}}^{[a,b]}\right)^{-1}(x) := 2\frac{x-a}{b-a} - 1, \quad x \in [a,b]$$

from $[a,b]$ to $[-1,1]$ and computing the s-th Lagrange polynomial on $[a,b]$ by means of

$$\mathcal{L}_s^{[a,b]}(x) := \mathcal{L}_s\left(\left(T_{\text{aff}}^{[a,b]}\right)^{-1}(x)\right).$$

This is equivalent to transforming the nodes x_s to $[a,b]$ via

$$x_s^{[a,b]} := T_{\text{aff}}^{[a,b]}(x_s) = a + (b-a)\frac{x_s+1}{2}.$$

The interpolation can be extended from an one-dimensional interval $[a,b]$ to an $(n+1)$-dimensional box $B_\nu = [a_1,b_1] \times \cdots \times [a_{n+1},b_{n+1}]$ by generating the tensor-product polynomials

$$\mathcal{L}_s^\nu(x) := \left(\mathcal{L}_{s_1}^{[a_1,b_1]} \otimes \cdots \otimes \mathcal{L}_{s_{n+1}}^{[a_{n+1},b_{n+1}]}\right)(x) = \prod_{i=1}^{n+1} \mathcal{L}_{s_i}^{[a_i,b_i]}(x_i)$$

with $s \in K := \{0,\ldots,p\}^{n+1} \subset \mathbb{N}_0^{n+1}$. This allows us to interpolate a function $f : B_\nu \to \mathbb{R}$ via

$$\left(\mathcal{I}_p^\nu[f]\right)(x) = \sum_{s\in K} f(x_s^\nu)\mathcal{L}_s^\nu(x),$$

where the x_s^ν correspond to the interpolation points of $\mathcal{L}_s^\nu$, i.e.

$$x_s^\nu := \left(x_{s_1}^{[a_1,b_1]}, \ldots, x_{s_{n+1}}^{[a_{n+1},b_{n+1}]}\right), \quad s \in K.$$

Remark 4.1. *If one of the cluster's dimensions is zero (i.e. $a_i = b_i$ for some $i \in \{1,\ldots,n+1\}$), our definition of interpolation polynomials does not make sense. Therefore, the interpolation polynomials in that direction are then simply replaced with a single constant function.*

We can extend the interpolation from above to functions that take *two* points in the $(n+1)$-dimensional space as arguments. In particular, this can be applied to the kernel function, resulting in

$$\left(\mathcal{I}_p^{\mu,\nu}[k]\right)(x,y) = \sum_{s,t\in K} k(x_s^\mu, x_t^\nu)\mathcal{L}_s^\mu(x)\mathcal{L}_t^\nu(y).$$

We can use this formula to approximate an entry of the system matrix, assuming that $\operatorname{supp}\phi_i \subset \mu$ and $\operatorname{supp}\phi_{i'} \subset \nu$:

$$\begin{aligned} A_{i,i'}^\phi = \langle A\phi_{i'}, \phi_i\rangle_{L^2(\Gamma)} &= \int_{\Gamma_N}\int_{\Gamma_N} \phi_{i'}(y)k(x,y)\phi_i(x)\mathrm{d}\sigma_x\mathrm{d}\sigma_y \\ &\approx \sum_{s,t\in K} k(x_s^\mu, x_t^\nu)\left(\int_{\Gamma_N}\phi_i(x)\mathcal{L}_s^\mu(x)\mathrm{d}\sigma_x\right)\left(\int_{\Gamma_N}\phi_{i'}(y)\mathcal{L}_t^\nu(y)\mathrm{d}\sigma_y\right). \end{aligned} \tag{4.17}$$

This approximation will of course only be valid when the kernel function is sufficiently smooth, which corresponds to the μ and ν being distant. We specify this in the following

Definition 4.2 (Admissibility condition)**.** *Two cluster μ and ν are said to satisfy the* admissibility condition with admissibility constant η, *if* $\max\{\operatorname{diam}(\mu),\operatorname{diam}(\nu)\} < \eta\operatorname{dist}(\mu,\nu)$.

Similar to Subsection 3.5, the actual implementation will make use of the stronger condition

$$\max\{\operatorname{diam}(B_\mu),\operatorname{diam}(B_\nu)\} < \eta\operatorname{dist}(B_\mu,B_\nu) \tag{4.18}$$

on the clusters' bounding boxes. Cluster pairs (and corresponding matrix blocks) that fulfill this condition will be called *admissible*.

In order to achieve our goal of providing a low-rank approximation of entire matrix blocks, we formalize the terms in equation (4.17):

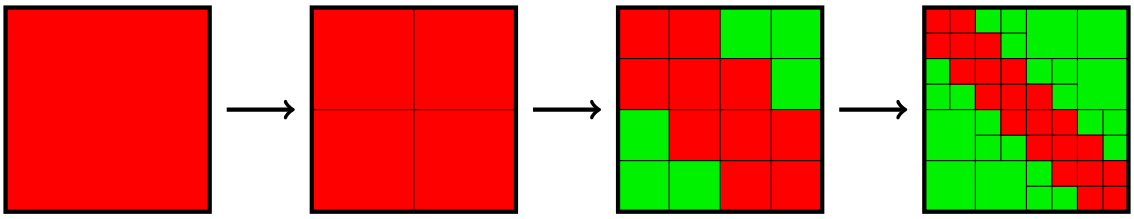

Figure 4.4: Hierarchical subdivision of a system matrix into admissible (green) and inadmissible (red) blocks.

Definition 4.3 (Cluster basis)**.** *Let T be a cluster tree. We call the family of matrices $\left(V_\phi^\mu\right)_{\mu\in T}$ with*

$$\left(V_\phi^\mu\right)_{i,s} := \int_{\Gamma_N} \phi_i(x)\mathcal{L}_s^\mu(x)\mathrm{d}\sigma_x, \quad i \in \mathbb{I}_\mu, s \in K \tag{4.19}$$

a cluster basis of T.

Note that the cluster basis is completely decoupled from the kernel function, which is taken care of in the following

Definition 4.4. *For admissible clusters μ and ν, the corresponding* coupling matrix *is defined as*

$$\left(S^{\mu,\nu}\right)_{s,t} := k(x_s^\mu, x_t^\nu), \quad s,t \in K.$$

We can now write the entire matrix block corresponding to the clusters μ and ν as $A_{\mu,\nu}^\phi \approx V_\phi^\mu S^{\mu,\nu}(V_\phi^\nu)^T$, whose storage requires $(\#\mu + \#\nu)\#K + \#K^2$ entries, which, for large clusters (i.e. $\#\mu, \#\nu \gg \#K$), is significantly lower than the original cost of $\#\mu\#\nu$. Moreover, the following proposition from [3, 4] allows us to control the error introduced by this approximation.

Proposition 4.5 (Error estimate)**.** *Let k satisfy the decay condition* (2.2)*. For two admissible clusters μ and ν as well as corresponding stable p-th order interpolation operators $\mathcal{I}_p^\mu$ and $\mathcal{I}_p^\nu$, the interpolation error for k is bounded by*

$$\left\|k - (\mathcal{I}_p^\mu \otimes \mathcal{I}_p^\nu)[k]\right\|_{L^\infty(B_\mu \times B_\nu)} \lesssim \mathcal{E}(p,\eta) := \frac{1}{\mathrm{dist}(B_\mu, B_\nu)^{2q+n}} \left(\frac{\eta}{\eta + C}\right)^p$$

for some constant C.

Remark 4.6. *As the lower bound for* $\mathrm{dist}(B_\mu, B_\nu)$ *scales with h_N for admissible clusters, p needs to be increased like* $\log(N)$ *in order to maintain sufficient accuracy. As our complexity estimates will depend on* $\#K = (p+1)^{n+1} \sim (\log(N))^{n+1}$*, the overall complexity of the algorithm will be linear-polylogarithmic.*

4.2. Determining nearfield and farfield

We have now obtained a low-rank approximation for large parts of the system matrix. Our next goal is to efficiently determine where this approximation is applicable, i.e. to cover a large part of the system matrix with admissible blocks.

On the one hand, the storage savings which can be obtained by the approximation improve with increasing size of the individual blocks. On the other hand, when the two cluster bounding boxes in the admissibility condition (4.18) differ significantly in size, the max-condition is quite inefficient. We will thus only consider admissible blocks where both clusters have the same level (and thus should be of similar size). Due to this reasoning, we obtain the list of admissible blocks by means of a recursive algorithm which is also illustrated in Figure 4.4: Starting with (Γ_N, Γ_N), the current cluster pair is checked for admissibility and, if admissible, added to a set L_{far} storing the farfield. Otherwise, the admissibility check will be performed on all possible pairs of son clusters for the two original clusters. When we arrive at a pair of inadmissible leaf clusters, it is added to the nearfield set L_{near}. This scheme provides us with a so-called *block cluster tree* [11].

Remark 4.7. *The low-rank approximation is only useful when its storage requirement* $((\#\mu + \#\nu)\#K + \#K^2)$ *is significantly smaller than* $\#\mu\#\nu$*, which corresponds to* $\#\mu, \#\nu \gg \#K$*. To achieve this, we require the leaf size to be of the same order of magnitude as* $\#K$*. In contrast,* $\#_{\text{leaf,max}} \sim m_{\tilde{d}}$ *was chosen for the Tausch-White wavelets (cf. Remark 3.8).*

Remark 4.8. *The block cluster structure of the nearfield makes it possible to store it in a block-sparse matrix as described in Remark 3.17.*

Thanks to the use of the admissibility condition, the size of the block cluster tree generated this way is bounded [8]:

Proposition 4.9. *The cardinality of a block cluster tree (and therefore its leaf count, too) constructed as described above is bounded by* $C(\eta, n)N$*. This also provides an upper bound for the algorithm's runtime.*

Each possible pair of simplices is contained in exactly one leaf block cluster, i.e.

$$\dot{\bigcup_{(\mu,\nu)\in L_{\text{near}}\cup L_{\text{far}}}} (\mu,\nu) = \{\pi_i : \pi_i \subset \Gamma_N\} \times \{\pi_i : \pi_i \subset \Gamma_N\}.$$

Due to the second property, the matrix blocks corresponding to the leaf block clusters cover the entire system matrix. Consequently, the following parts are already sufficient for a complete approximation:

1. The cluster basis V_ϕ^μ for all clusters $\mu \in T$,
2. the nearfield blocks $A_{\mu,\nu}^\phi$ for all cluster blocks $(\mu,\nu) \in L_{\text{near}}$ and
3. the coupling matrices $S^{\mu,\nu}$ for all cluster blocks $(\mu,\nu) \in L_{\text{far}}$.

These coefficients will be computed once and reused for every application of the matrix-vector product.

Remark 4.10. *Every coupling matrix consists of* $\#K^2 = (p+1)^{2(n+1)}$ *entries. For practical applications, values in the range of* $p = 5$ *are not unusual and lead to large values of* $\#K^2$*. The whole method's memory requirements are therefore dominated by the coupling matrices.*

4.3. Computing the matrix-vector product

We will now use the coefficients obtained in the previous subsections to actually approximate the matrix-vector product. Using the block cluster structure from above, we can write it as

$$y^\phi = A^\phi x^\phi \approx (A_{\text{near}}^\phi + A_{\text{far}}^\phi)x^\phi = \sum_{(\mu,\nu)\in L_{\text{near}}} A_{\mu,\nu}^\phi x_\nu^\phi + \sum_{(\mu,\nu)\in L_{\text{far}}} V_\phi^\mu S^{\mu,\nu} \left(V_\phi^\nu\right)^T x_\nu^\phi, \tag{4.20}$$

where the products $A_{\mu,\nu}^\phi x_\nu^\phi$ and $V_\phi^\mu S^{\mu,\nu}\left(V_\phi^\nu\right)^T x_\nu^\phi$ are meant to have the same index structure as y_μ^ϕ, i.e. the corresponding addition is performed as e.g. $y_\mu^\phi := y_\mu^\phi + A_{\mu,\nu}^\phi x_\nu^\phi$. This approach can be further optimized by eliminating redundant computations: A naive approach will compute the product $\left(V_\phi^\nu\right)^T x_\nu^\phi$ once *per block cluster* rather than just once per cluster. Similarly, one can optimize the multiplication with V_ϕ^μ to obtain

$$\sum_{(\mu,\nu)\in L_{\text{far}}} V_\phi^\mu S^{\mu,\nu} \left(V_\phi^\nu\right)^T x_\nu^\phi = \sum_{\mu\in T} V_\phi^\mu \left(\sum_{(\mu,\nu)\in L_{\text{far}}} S^{\mu,\nu} \left(V_\phi^\nu\right)^T x_\nu^\phi \right).$$

This corresponds to computing the matrix-vector product in four distinct steps:

1. *Forward transformation:* Compute $x^\nu := (V_\phi^\nu)^T x_\nu^\phi$ for each cluster $\nu \in T$.
2. *Coupling:* Apply the coupling matrices by means of $y^\mu := \sum_{(\mu,\nu)\in L_{\text{far}}} S^{\mu,\nu} x^\nu$.
3. *Backward transformation:* Initialize y^ϕ with zeroes and perform the addition $y_\mu^\phi := y_\mu^\phi + V_\phi^\mu y^\mu$ for each cluster $\mu \in T$.
4. *Nearfield:* For each leaf cluster μ, add the corresponding nearfield blocks: $y_\mu^\phi := y_\mu^\phi + \sum_{(\mu,\nu)\in L_{\text{near}}} A_{\mu,\nu}^\phi x_\nu^\phi$.

Remark 4.11. *The vectors x^ν and y^μ are* not *just re-indexed versions of x^ϕ and y^ϕ, but rather individual temporary vectors for every single cluster. They each contain $\#K$ entries.*

As both the nearfield and the farfield consist of a linear number of block clusters, steps 2 and 4 can be performed with linear complexity [4]). In contrast, steps 1 and 3 operate on *all* basis functions for *each* level, leading to linear-logarithmic cost. The upcoming subsection will be concerned with eliminating this remaining logarithm.

4.4. Nested cluster basis

The cause for the linear-logarithmic cost in the matrix-vector product is the structure of the cluster basis: the row count of a particular matrix of the cluster basis is equal to the corresponding cluster's cardinality. Therefore, computing the interactions for all clusters on a single level will already require $\mathcal{O}(N)$ interactions, resulting in $\mathcal{O}(N \log(N))$ computations overall.

This obstacle can be overcome by harnessing the recursive nature of the cluster structure. Remember that the cluster basis was constructed on the Lagrange interpolation polynomials up to a particular degree p. Although the interpolation nodes differ between clusters, their span is always identical to the $(n+1)$-fold tensor product of the space of one-dimensional polynomials of order p:

$$\operatorname{span}\left\{\mathcal{L}_s^\mu : s \in K\right\} = \operatorname{span}\left\{\mathcal{L}_t^\nu : t \in K\right\} \equiv \mathbb{P}_p^{n+1}$$

In particular, the Lagrange polynomials from one cluster μ can be written as linear combinations of the Lagrange polynomials from any other cluster ν:

$$\mathcal{L}_s^\mu = \sum_{t' \in K} T_{t',s}^{\nu,\mu} \mathcal{L}_{t'}^\nu, \quad s \in K. \tag{4.21}$$

To obtain actual values for these translation coefficients, we repeat the Lagrange polynomial's interpolation property:

$$\mathcal{L}_{t'}^\nu(x_t^\nu) = \delta_{t't}, \quad t, t' \in K. \tag{4.22}$$

Evaluating both sides of equation (4.21) at x_t^ν and inserting the identity (4.22) into the right hand side provides us with

$$\mathcal{L}_s^\mu(x_t^\nu) = \sum_{t' \in K} T_{t',s}^{\nu,\mu} \mathcal{L}_{t'}^\nu(x_t^\nu) = \sum_{t' \in K} T_{t',s}^{\nu,\mu} \delta_{t't} = T_{t,s}^{\nu,\mu}, \quad s, t \in K.$$

Equation (4.19) for a cluster μ can then be rewritten in terms of the entries of a son cluster μ' as:

$$\left(V_\phi^\mu\right)_{i,s} = \int_{\Gamma_N} \phi_i(x) \mathcal{L}_s^\mu(x) \mathrm{d}\sigma_x = \sum_{t' \in K} T_{t',s}^{\mu',\mu} \int_{\Gamma_N} \phi_i(x) \mathcal{L}_{t'}^{\mu'}(x) \mathrm{d}\sigma_x = \left(V_\phi^{\mu'} T^{\mu',\mu}\right)_{i,s} \tag{4.23}$$

for $s \in K$, $i \in \mathbb{I}_{\mu'} \subset \mathbb{I}_\mu$. Hence, V_ϕ^μ can be constructed in its entirety from the son clusters $\mu_{\mathrm{son},1}, \ldots, \mu_{\mathrm{son},\ell_\mu}$ via

$$V_\phi^\mu = \begin{bmatrix} V_\phi^{\mu_{\mathrm{son},1}} T^{\mu_{\mathrm{son},1},\mu} \\ \vdots \\ V_\phi^{\mu_{\mathrm{son},\ell_\mu}} T^{\mu_{\mathrm{son},\ell_\mu},\mu} \end{bmatrix}. \tag{4.24}$$

It is thus possible to discard the cluster basis matrices for non-leaf clusters and store the translation matrices $T^{\mu_{\mathrm{son}},\mu}$ for $\mu_{\mathrm{son}} \prec \mu$ instead to achieve a reduction in storage cost from $\#\mu\#K$ to $\ell_\mu \#K^2$. The cluster basis of a non-leaf cluster can still be reconstructed recursively from its son clusters. We formalize this in the following

Definition 4.12 (Nested cluster basis)**.** *Let T be a cluster tree. We call a corresponding cluster basis $\left(V_\phi^\mu\right)_{\mu \in T}$* nested*, if for each non-leaf cluster μ with son clusters $\mu_{\mathrm{son},i}$ transfer matrices $T^{\mu_{\mathrm{son},i},\mu} \in \mathbb{R}^{\#K \times \#K}$ exist such that equation* (4.24) *is satisfied.*

We can also use this structure to reduce the complexity of the matrix-vector product: The steps with linear-logarithmic asymptotic complexity were the first, where $x^\nu := \left(V_\phi^\nu\right)^T x_\nu^\phi$ is computed for each cluster $\nu \in T$ and the third, which involves the operation $y_\mu^\phi := y_\mu^\phi + V_\phi^\mu y^\mu$ for each cluster $\mu \in T$. Using the representation (4.24), the first step can be rewritten for non-leaf clusters ν as

$$x^\nu = \begin{bmatrix} V_\phi^{\nu_{\text{son},1}} T^{\nu_{\text{son},1},\nu} \\ \vdots \\ V_\phi^{\nu_{\text{son},\ell_\nu}} T^{\nu_{\text{son},\ell_\nu},\nu} \end{bmatrix}^T \begin{bmatrix} x_{\nu_{\text{son},1}}^\phi \\ \vdots \\ x_{\nu_{\text{son},\ell_\nu}}^\phi \end{bmatrix} = \sum_{i=1}^{\ell_\nu} \left(T^{\nu_{\text{son},i},\nu}\right)^T \left(V_\phi^{\nu_{\text{son},i}}\right)^T x_{\nu_{\text{son},i}}^\phi = \sum_{i=1}^{\ell_\nu} \left(T^{\nu_{\text{son},i},\nu}\right)^T x^{\nu_{\text{son},i}}.$$

This allows us to compute the coefficient vectors x^ν directly for all leaf clusters and then use those coefficients to recursively compute the father cluster's coefficient vectors. For each non-leaf ν cluster only a constant number of operations is required (namely ℓ_ν multiplications of $(\#K \times \#K)$-matrices and $\#K$-row vectors), resulting in a total runtime and memory cost of $\mathcal{O}(N\#K)$ [12] (provided $\#_{\text{leaf,max}} \sim \#K$, resulting in a total number of $\sim \frac{N}{\#K}$ clusters). This justifies the name *fast forward transform*.

Similarly, performing the operation $y_\mu^\phi := y_\mu^\phi + V_\phi^\mu y^\mu$ for all clusters $\mu \in T$ is identical to just computing $y_{\mu^l}^\phi = \sum_{\mu \in T:\mu^l \subset \mu} \left(V_\phi^\mu y^\mu\right)_{\mu^l}$ for all *leaf* clusters $\mu^l \in L(T)$, where $\left(V_\phi^\mu y^\mu\right)_{\mu^l}$ denotes the restriction of $V_\phi^\mu y^\mu$ to the entries corresponding to simplices from μ^l. For a cluster $\nu \supset \mu^l$ (and corresponding son cluster $\nu_{\text{son}} \prec \nu$), this sum can be split into

$$\begin{aligned} y_{\mu^l}^\phi &= \sum_{\mu \in T \setminus \{\nu\}:\mu^l \subset \mu} \left(V_\phi^\mu y^\mu\right)_{\mu^l} + \left(V_\phi^\nu y^\nu\right)_{\mu^l} \\ &= \sum_{\mu \in T \setminus \{\nu\}:\mu^l \subset \mu} \left(V_\phi^\mu y^\mu\right)_{\mu^l} + \left(V_\phi^{\nu_{\text{son}}} T^{\nu_{\text{son}},\nu} y^\nu\right)_{\mu^l} \\ &= \sum_{\mu \in T \setminus \{\nu,\nu_{\text{son}}\}:\mu^l \subset \mu} \left(V_\phi^\mu y^\mu\right)_{\mu^l} + \left(V_\phi^{\nu_{\text{son}}} (y^{\nu_{\text{son}}} + T^{\nu_{\text{son}},\nu} y^\nu)\right)_{\mu^l}. \end{aligned}$$

This formulation gives rise to the recursive approach of the *fast backward transform*, which is essentially a reverse fast forward transform (and therefore has the same asymptotic runtime). For a non-leaf cluster ν, the products $T^{\nu_{\text{son}},\nu} y^\nu$ can be computed for its son clusters ν_{son} and the results are added to the corresponding vectors $y^{\nu_{\text{son}}}$. The recursion can be started at the root cluster and continued down to the leaf clusters. It is then sufficient to compute $y_{\mu^l}^\phi = V_\phi^{\mu^l} y^{\mu^l}$ for all leaf clusters $\mu^l \in L(T)$, where the vectors y^{μ^l} contain their father cluster's coefficient vectors as described above.

5. The $\mathcal{H}^2$-Wavelet Method

We shall now introduce a method for directly setting up the compressed system matrix in the multi-scale basis. With a naive approach, matrix blocks corresponding to coarser-scale wavelets would require the computation of all corresponding single-scale entries. We will thus employ the far-field approximation presented in the previous section to directly compute those blocks in the multi-scale basis.

The asymptotic complexity estimates in the following subsections can only be maintained under the condition that the maximum number of scaling functions and wavelets provided by a particular cluster are each bounded by a constant C_Q. For a wavelet-optimized leaf size $\#_{\text{leaf,max}} \sim m_{\tilde{d}}$, this is true (cf. Remark 3.7). For our numerical experiments, however, we will choose $\#_{\text{leaf,max}} \sim \#K$ (cf. Subsection 6.1.1). This leads to $C_Q \sim \#K$ (and thus introduces additional logarithms due to $\#K = (p+1)^{n+1} \sim (\log(N))^{n+1}$), but decreases the overall computation time as there are significantly fewer cluster pairs for which the $(\#K \times \#K)$-coupling matrices need to be applied. It also introduces more non-negligible matrix entries, but we will see in Section 6 that the algorithm's memory usage is dominated by the size of the nearfield, anyway.

5.1. Approximation of admissible blocks

Recall that the system matrix in the multi-scale basis consists of entries of the form $A^{\Psi}_{k,k'} = \langle A\vartheta_{k'}, \vartheta_k\rangle$ with $k, k' \in \mathbb{I}^{\Psi}_N$ where ϑ_k and $\vartheta_{k'}$ are wavelets or scaling functions of the root cluster, respectively. For our further considerations, we require a fast method to determine these entries. Note that we will perform the following calculations exemplarily for the wavelet-wavelet case $\vartheta_k \equiv \psi^{\mu}_{j,k}$ and $\vartheta_{k'} \equiv \psi^{\nu}_{j',k'}$. The other cases (where ϑ_k and/or $\vartheta_{k'}$ are scaling functions rather than wavelets) can be treated analogously.

In Subsection 4.1, equation (4.17) lead us to the approximation

$$A^{\phi}_{i,i'} = \langle A\phi_{i'}, \phi_i\rangle_{L^2(\Gamma)} \approx \sum_{s,t\in K} k(x^{\mu}_s, x^{\nu}_t) \left(\int_{\Gamma_N} \phi_i(x)\mathcal{L}^{\mu}_s(x)\mathrm{d}\sigma_x\right)\left(\int_{\Gamma_N} \phi_{i'}(y)\mathcal{L}^{\nu}_t(y)\mathrm{d}\sigma_y\right)$$

for admissible clusters $\mu \supset \operatorname{supp}(\phi_i)$ and $\nu \supset \operatorname{supp}(\phi_{i'})$. Extending this approximation to the wavelets (which also fulfill $\operatorname{supp}(\psi^{\mu}_{j,k}) \subset \mu$ and $\operatorname{supp}(\psi^{\nu}_{j',k'}) \subset \nu$) yields

$$A^{\Psi}_{k,k'} = \langle A\psi^{\nu}_{j',k'}, \psi^{\mu}_{j,k}\rangle \approx \sum_{s,t\in K} k(x^{\mu}_s, x^{\nu}_t) \left(\int_{\Gamma_N} \psi^{\mu}_{j,k}(x)\mathcal{L}^{\mu}_s(x)\mathrm{d}\sigma_x\right)\left(\int_{\Gamma_N} \psi^{\nu}_{j',k'}(y)\mathcal{L}^{\nu}_t(y)\mathrm{d}\sigma_y\right).$$

With this representation, we can avoid setting up the single-scale matrix block $A^{\phi}_{\mu,\nu}$ in order to compute the entry $A^{\Psi}_{k,k'}$. To this end, we need the following

Definition 5.1 (Multi-scale cluster basis)**.** *Let T be a cluster tree and $\mathbf{\Phi}_N$ and $\mathbf{\Psi}_N$ the sets of all scaling functions and wavelets, respectively. The* scaling function and wavelet cluster bases *are the families $(V^{\mu}_{\Phi})_{\mu\in T}$ and $(V^{\mu}_{\Psi})_{\mu\in T}$ of matrices defined via*

$$(V^{\mu}_{\Phi})_{k,s} := \int_{\Gamma_N} \varphi^{\mu}_{j,k}(x)\mathcal{L}^{\mu}_s(x)\mathrm{d}\sigma_x, \quad k \in \mathbb{I}^{\Phi}_{\mu}, s \in K,$$
$$(V^{\mu}_{\Psi})_{k,s} := \int_{\Gamma_N} \psi^{\mu}_{j,k}(x)\mathcal{L}^{\mu}_s(x)\mathrm{d}\sigma_x, \quad k \in \mathbb{I}^{\Psi}_{\mu}, s \in K.$$

Their combination is called multi-scale cluster basis*.*

Note that this formulation is very similar to Definition 4.3, but for a different choice of ansatz functions. Hence, with the scaling function basis, we can write

$$\tilde{A}^{\Phi,\Phi}_{\mu,\nu} := V^{\mu}_{\Phi} S^{\mu,\nu} \left(V^{\nu}_{\Phi}\right)^T, \quad \tilde{A}^{\Phi,\Psi}_{\mu,\nu} := V^{\mu}_{\Phi} S^{\mu,\nu} \left(V^{\nu}_{\Psi}\right)^T, \quad \tilde{A}^{\Psi,\Phi}_{\mu,\nu} := V^{\mu}_{\Psi} S^{\mu,\nu} \left(V^{\nu}_{\Phi}\right)^T,$$

or, more general,

$$\tilde{A}^{\Upsilon,\Upsilon'}_{\mu,\nu} := V^{\mu}_{\Upsilon} S^{\mu,\nu} \left(V^{\nu}_{\Upsilon'}\right)^T \tag{5.25}$$

with $\Upsilon, \Upsilon' \in \{\Phi, \Psi\}$.

Lemma 5.2. *Let k satisfy the decay condition* (2.2) *and (μ, ν) be a pair of admissible clusters on a 2^n-tree with levels j and j'. For corresponding stable p-th order interpolation operators $\mathcal{I}^{\mu}_p$ and $\mathcal{I}^{\nu}_p$, the error of a single approximated matrix entry is bounded by*

$$\left|A^{\Upsilon,\Upsilon'}_{k,k'} - \tilde{A}^{\Upsilon,\Upsilon'}_{k,k'}\right| \lesssim \frac{2^{-(j+j')\frac{n}{2}}}{\operatorname{dist}(B_{\mu}, B_{\nu})^{2q+n}} \left(\frac{\eta}{\eta + C}\right)^p, \quad \Upsilon, \Upsilon' \in \{\Phi, \Psi\}$$

for some constant $C > 0$.

Proof. Without loss of generality let $\Upsilon = \Upsilon' = \Psi$. This specialization is valid because the remainder of the proof does not rely on any special properties (i.e. vanishing moments or orthogonality) of wavelets as opposed to scaling functions.

Writing the approximated kernel function as $\tilde{k} := \left(\mathcal{I}_p^\mu \otimes \mathcal{I}_p^\nu\right)[k]$, we obtain the following representation of the error:

$$\left|A_{k,k'}^{\Psi,\Psi'} - \tilde{A}_{k,k'}^{\Psi,\Psi'}\right| = \left|\int_{\Gamma_N}\int_{\Gamma_N} \psi_{j,k}^\mu(x)\psi_{j',k'}^\nu(y)[k(x,y) - \tilde{k}(x,y)]\mathrm{d}\sigma_y\mathrm{d}\sigma_x\right|.$$

For a more fine-grained control of the error, we insert the coefficient representation (3.14) to get

$$\begin{aligned}\left|A_{k,k'}^{\Psi,\Psi'} - \tilde{A}_{k,k'}^{\Psi,\Psi'}\right| &= \left|\int_{\Gamma_N}\int_{\Gamma_N} \sum_{i=1}^N \omega_i^{j,k}\phi_i(x) \sum_{i'=1}^N \omega_{i'}^{j',k'}\phi_{i'}(y)[k(x,y) - \tilde{k}(x,y)]\mathrm{d}\sigma_y\mathrm{d}\sigma_x\right| \\ &= \left|\sum_{i,i'=1}^N \frac{\omega_i^{j,k}\omega_i^{j',k'}}{\sqrt{|\pi_i||\pi_{i'}|}}\int_{\pi_i}\int_{\pi_{i'}} [k(x,y) - \tilde{k}(x,y)]\mathrm{d}\sigma_y\mathrm{d}\sigma_x\right|.\end{aligned}$$

We can now use the estimate from Proposition 4.5 for another simplification:

$$\left|A_{k,k'}^{\Psi,\Psi'} - \tilde{A}_{k,k'}^{\Psi,\Psi'}\right| \lesssim \sum_{i,i'=1}^N \frac{|\omega_i^{j,k}||\omega_i^{j',k'}|}{\sqrt{|\pi_i||\pi_{i'}|}}\int_{\pi_i}\int_{\pi_{i'}} \mathcal{E}(p,\eta)\mathrm{d}\sigma_y\mathrm{d}\sigma_x = \sum_{i,i'=1}^N |\omega_i^{j,k}||\omega_i^{j',k'}|\sqrt{|\pi_i||\pi_{i'}|}\mathcal{E}(p,\eta).$$

To conclude this part of the proof, we use Lemma 3.12 and observe

$$2^{-j\frac{n}{2}} \gtrsim \int_{\Gamma_N} |\psi_{j,k}^\nu(x)|\mathrm{d}\sigma_x = \sum_{i=1}^N |\omega_i^{j,k}|\int_{\pi_i}|\phi_i(x)|\mathrm{d}\sigma_x = \sum_{i=1}^N |\omega_i^{j,k}|\sqrt{|\pi_i|}.$$

Inserting this estimate yields

$$\left|A_{k,k'}^{\Psi,\Psi'} - \tilde{A}_{k,k'}^{\Psi,\Psi'}\right| \lesssim 2^{-(j+j')\frac{n}{2}}\mathcal{E}(p,\eta) = \frac{2^{-(j+j')\frac{n}{2}}}{\operatorname{dist}(B_\mu,B_\nu)^{2q+n}}\left(\frac{\eta}{\eta+C}\right)^p.$$

□

As each matrix block has at most C_Q rows and columns, we can state the following

Corollary 5.3 (Error estimate for matrix blocks)**.** *Under the conditions of Lemma 5.2, the error of an approximated matrix block is bounded by*

$$\left\|A_{\mu,\nu}^{\Upsilon,\Upsilon'} - \tilde{A}_{\mu,\nu}^{\Upsilon,\Upsilon'}\right\|_2 \lesssim C_Q\frac{2^{-(j+j')\frac{n}{2}}}{\operatorname{dist}(B_\mu,B_\nu)^{2q+n}}\left(\frac{\eta}{\eta+C}\right)^p, \quad \Upsilon,\Upsilon' \in \{\Phi,\Psi\},$$

with the same constant $C > 0$ as in Lemma 5.2.

We will now investigate the cost of approximating a single block:

Lemma 5.4. *For given coupling matrices as well as multi-scale cluster basis matrices, a single admissible block $A_{\mu,\nu}^{\Upsilon,\Upsilon'}$ can be approximated with $\mathcal{O}(\#K^2)$ operations.*

Proof. The desired approximation consists of computing the matrix products $\tilde{A}_{\mu,\nu}^{\Upsilon,\Upsilon'} := \left(V_\Upsilon^\mu S^{\mu,\nu}\right)\left(V_{\Upsilon'}^\nu\right)^T$, where both V_Υ^μ and $V_{\Upsilon'}^\nu$ will each have exactly $\#K$ columns and at most C_Q rows and $S^{\mu,\nu}$ has exactly $\#K$ rows and columns. Therefore the effort of computing $V_\Upsilon^\mu S^{\mu,\nu}$ can be bounded by $\mathcal{O}(C_Q\#K^2)$, while the subsequent multiplication with $\left(V_{\Upsilon'}^\nu\right)^T$ requires an additional $\mathcal{O}(C_Q^2\#K)$ operations. As C_Q is considered to be constant, we end up with a total effort of $\mathcal{O}(\#K^2)$. □

Remark 5.5 (Cost reduction)**.** *Note that a single block $\tilde{A}_{\mu,\nu}^{\Upsilon,\Upsilon'}$ only consists of at most C_Q^2 entries, anyway. The approximation* (5.25) *is nevertheless reasonable, as it avoids setting up (and transforming) the entire single-scale block $A_{\mu,\nu}^\phi$, which would take $\mathcal{O}(\#\mu\#\nu)$ operations.*

Keep in mind that these approximations are only valid for *admissible* cluster pairs (μ,ν)! In particular, the pair (Γ_N,Γ_N) of root clusters will not be admissible. We will therefore have to develop a more sophisticated method to determine *arbitrary* matrix blocks later.

5.2. The multi-scale cluster basis

In order to take advantage of the possible effort reduction of the approximation technique presented in the previous subsection, we still need an efficient way to determine the multi-scale cluster basis. For the sake of brevity, we will only consider the wavelet cluster basis in this subsection, but the scaling function cluster basis can be handled identically.

Computing a single entry $\left(V_\Psi^\mu\right)_{k,s}$ could be performed by making use of the wavelet's representation (3.14) in the single-scale basis:

$$\left(V_\Psi^\mu\right)_{k,s} = \int_{\Gamma_N} \psi_{j,k}^\mu(x)\mathcal{L}_s^\mu(x)\mathrm{d}\sigma_x = \sum_{i\in\mathbb{I}_{j,k}} \omega_i^{j,k} \int_{\Gamma_N} \phi_i(x)\mathcal{L}_s^\mu(x)\mathrm{d}\sigma_x = \sum_{i\in\mathbb{I}_{j,k}} \omega_i^{j,k}\left(V_\phi^\mu\right)_{i,s}.$$

This approach would require setting up the regular cluster basis (as opposed to the *nested* cluster basis), however, which also would still need to be transformed to the multi-scale basis.

Fortunately, the hierarchical structures of the nested cluster basis and the wavelet basis are identical, which makes it possible to combine both approaches in a recursive algorithm that computes the wavelet cluster basis. We begin by making use of the refinement relation (3.8):

$$\begin{aligned} V_\Psi^\mu &= \left[\int_{\Gamma_N} \left(\boldsymbol{\Psi}_j^\mu\right)^T(x)\mathcal{L}_s^\mu(x)\mathrm{d}\sigma_x\right]_{s\in K} = \left[\int_{\Gamma_N} \left(\boldsymbol{\Phi}_{J+1}^\mu Q_{J,\Psi}^\mu\right)^T(x)\mathcal{L}_s^\mu(x)\mathrm{d}\sigma_x\right]_{s\in K} \\ &= \left(Q_{J,\Psi}^\mu\right)^T\left[\int_{\Gamma_N} \left(\boldsymbol{\Phi}_{J+1}^\mu\right)^T(x)\mathcal{L}_s^\mu(x)\mathrm{d}\sigma_x\right]_{s\in K} = \left(Q_{J,\Psi}^\mu\right)^T V_\phi^\mu, \end{aligned}$$

as $\boldsymbol{\Phi}_{J+1}^\mu := \{\phi_i : i \in \mathbb{I}_\mu\}$ consists of the single-scale basis functions corresponding to μ. For a non-leaf cluster, we obtain

$$\begin{aligned} V_\Psi^\mu &= \left(Q_{j,\Psi}^\mu\right)^T\left[\int_{\Gamma_N} \left(\boldsymbol{\Phi}_{j+1}^\mu\right)^T(x)\mathcal{L}_s^\mu(x)\mathrm{d}\sigma_x\right]_{s\in K} = \left(Q_{j,\Psi}^\mu\right)^T \begin{bmatrix} \int_{\Gamma_N} \left(\boldsymbol{\Phi}_{j+1}^{\mu_{\mathrm{son},1}}\right)^T(x)\mathcal{L}_s^\mu(x)\mathrm{d}\sigma_x \\ \vdots \\ \int_{\Gamma_N} \left(\boldsymbol{\Phi}_{j+1}^{\mu_{\mathrm{son},\ell_\mu}}\right)^T(x)\mathcal{L}_s^\mu(x)\mathrm{d}\sigma_x \end{bmatrix}_{s\in K} \\ &\overset{(4.21)}{=} \left(Q_{j,\Psi}^\mu\right)^T \begin{bmatrix} \int_{\Gamma_N} \left(\boldsymbol{\Phi}_{j+1}^{\mu_{\mathrm{son},1}}\right)^T(x)\sum_{t'\in K} T_{t',s}^{\mu_{\mathrm{son},1},\mu}\mathcal{L}_{t'}^{\mu_{\mathrm{son},1}}(x)\mathrm{d}\sigma_x \\ \vdots \\ \int_{\Gamma_N} \left(\boldsymbol{\Phi}_{j+1}^{\mu_{\mathrm{son},\ell_\mu}}\right)^T(x)\sum_{t'\in K} T_{t',s}^{\mu_{\mathrm{son},\ell_\mu},\mu}\mathcal{L}_{t'}^{\mu_{\mathrm{son},\ell_\mu}}(x)\mathrm{d}\sigma_x \end{bmatrix}_{s\in K} \\ &\overset{(4.23)}{=} \left(Q_{j,\Psi}^\mu\right)^T \begin{bmatrix} \sum_{t'\in K} T_{t',s}^{\mu_{\mathrm{son},1},\mu}\int_{\Gamma_N} \left(\boldsymbol{\Phi}_{j+1}^{\mu_{\mathrm{son},1}}\right)^T(x)\mathcal{L}_{t'}^{\mu_{\mathrm{son},1}}(x)\mathrm{d}\sigma_x \\ \vdots \\ \sum_{t'\in K} T_{t',s}^{\mu_{\mathrm{son},\ell_\mu},\mu}\int_{\Gamma_N} \left(\boldsymbol{\Phi}_{j+1}^{\mu_{\mathrm{son},\ell_\mu}}\right)^T(x)\mathcal{L}_{t'}^{\mu_{\mathrm{son},\ell_\mu}}(x)\mathrm{d}\sigma_x \end{bmatrix}_{s\in K} \overset{(4.24)}{=} \left(Q_{j,\Psi}^\mu\right)^T \begin{bmatrix} V_\Phi^{\mu_{\mathrm{son},1}} T^{\mu_{\mathrm{son},1},\mu} \\ \vdots \\ V_\Phi^{\mu_{\mathrm{son},\ell_\mu}} T^{\mu_{\mathrm{son},\ell_\mu},\mu} \end{bmatrix}. \end{aligned}$$

Therefore, the multi-scale cluster basis can be computed recursively by simply transforming the corresponding son clusters' scaling function bases in $\mathcal{O}(N\#K^2)$ operations. It has storage cost $\mathcal{O}(N\#K)$.

5.3. Recursive computation of inadmissible matrix blocks

As mentioned before, we have yet to provide a fast method for approximating *inadmissible* blocks of the multi-scale matrix. In that case, we can employ the refinement relation (3.8) write a block $A_{\mu,\nu}^{\Upsilon,\Upsilon'}$ ($\Upsilon,\Upsilon' \in \{\Phi,\Psi\}$) in terms of the corresponding son clusters' blocks:

$$\begin{aligned} A_{\mu,\nu}^{\Upsilon,\Upsilon'} = \left[\langle A(\boldsymbol{\Upsilon}'^\nu)^T, \boldsymbol{\Upsilon}^\mu\rangle\right] &= \left(Q_\Upsilon^\mu\right)^T \begin{bmatrix} \langle A(\boldsymbol{\Phi}^{\nu_{\mathrm{son},1}})^T, \boldsymbol{\Phi}^{\mu_{\mathrm{son},1}}\rangle & \cdots & \langle A(\boldsymbol{\Phi}^{\nu_{\mathrm{son},\ell_\nu}})^T, \boldsymbol{\Phi}^{\mu_{\mathrm{son},1}}\rangle \\ \vdots & & \vdots \\ \langle A(\boldsymbol{\Phi}^{\nu_{\mathrm{son},1}})^T, \boldsymbol{\Phi}^{\mu_{\mathrm{son},\ell_\mu}}\rangle & \cdots & \langle A(\boldsymbol{\Phi}^{\nu_{\mathrm{son},\ell_\nu}})^T, \boldsymbol{\Phi}^{\mu_{\mathrm{son},\ell_\mu}}\rangle \end{bmatrix} Q_{\Upsilon'}^\nu \\ &= \left(Q_\Upsilon^\mu\right)^T \begin{bmatrix} A_{\mu_{\mathrm{son},1},\nu_{\mathrm{son},1}}^{\Phi,\Phi} & \cdots & A_{\mu_{\mathrm{son},1},\nu_{\mathrm{son},\ell_\nu}}^{\Phi,\Phi} \\ \vdots & & \vdots \\ A_{\mu_{\mathrm{son},\ell_\mu},\nu_{\mathrm{son},1}}^{\Phi,\Phi} & \cdots & A_{\mu_{\mathrm{son},\ell_\mu},\nu_{\mathrm{son},\ell_\nu}}^{\Phi,\Phi} \end{bmatrix} Q_{\Upsilon'}^\nu, \end{aligned} \tag{5.26}$$

provided both μ and ν are non-leaf clusters.

Remark 5.6. *As these considerations are not dependent of any levels, we will omit the level index j in the following.*

All admissible blocks for the son clusters can then be approximated by means of (5.25), while the subdivision is repeated recursively for inadmissible blocks until one or both clusters are leaves. If only one of the clusters is a leaf, the subdivision process can be continued with

$$A_{\mu,\nu}^{\Upsilon,\Upsilon'} = \left[A_{\mu,\nu_{\text{son},1}}^{\Upsilon,\Phi}, \ldots, A_{\mu,\nu_{\text{son},\ell_\nu}}^{\Upsilon,\Phi} \right] Q_{\Upsilon'}^{\nu} \tag{5.27}$$

in the case of ν being a non-leaf cluster, or with

$$A_{\mu,\nu}^{\Upsilon,\Upsilon'} = \left(Q_{\Upsilon}^{\mu}\right)^T \begin{bmatrix} A_{\mu_{\text{son},1},\nu}^{\Phi,\Upsilon'} \\ \vdots \\ A_{\mu_{\text{son},\ell_\mu},\nu}^{\Phi,\Upsilon'} \end{bmatrix} \tag{5.28}$$

for a non-leaf cluster μ. In the case of two inadmissible leaf clusters, (μ,ν) will belong to the nearfield. It is then possible to simply transform the corresponding single-scale nearfield block:

$$A_{\mu,\nu}^{\Upsilon,\Upsilon'} = \left(Q_{\Upsilon}^{\mu}\right)^T A_{\mu,\nu}^{\phi} Q_{\Upsilon'}^{\nu}. \tag{5.29}$$

Note that, in contrast to the $\mathcal{H}^2$-method, the multi-scale matrix also contains blocks where the corresponding clusters' levels differ. In this case, we refrain from immediately employing the symmetric subdivision (5.26) but rather only subdivide ν as in (5.27) (for $j_\mu > j_\nu$) or μ via (5.28) (for $j_\mu < j_\nu$). These thoughts are condensed in the following function:

Function recursivelyDetermineMatrixBlock(μ, ν, Υ, Υ')

Result: Approximation of the block $A_{\mu,\nu}^{\Upsilon,\Upsilon'}$.
begin
  if (μ,ν) *is admissible* **then**
    return $V_{\Upsilon}^{\mu} S^{\mu,\nu} \left(V_{\Upsilon'}^{\nu}\right)^T$;
  else if μ *and* ν *are leaf clusters* **then**
    return $\left(Q_{\Upsilon}^{\mu}\right)^T A_{\mu,\nu}^{\phi} Q_{\Upsilon'}^{\nu}$;
  else if ν *is not a leaf cluster and (μ is a leaf cluster or $j_\mu > j_\nu$)* **then**
    for $\nu_{\text{son}} \prec \nu$ **do**
      compute $A_{\mu,\nu_{\text{son}}}^{\Upsilon,\Phi}$:= `recursivelyDetermineMatrixBlock`$(\mu, \nu_{\text{son}}, \Upsilon, \Phi)$;
    return $\left[A_{\mu,\nu_{\text{son},1}}^{\Upsilon,\Phi}, \ldots, A_{\mu,\nu_{\text{son},\ell_\nu}}^{\Upsilon,\Phi} \right] Q_{\Upsilon'}^{\nu}$;
  else if μ *is not a leaf cluster and (ν is a leaf cluster or $j_\mu < j_\nu$)* **then**
    for $\mu_{\text{son}} \prec \mu$ **do**
      compute $A_{\mu_{\text{son}},\nu}^{\Phi,\Upsilon'}$:= `recursivelyDetermineMatrixBlock`$(\mu_{\text{son}}, \nu, \Phi, \Upsilon')$;
    return $\left(Q_{\Upsilon}^{\mu}\right)^T \begin{bmatrix} A_{\mu_{\text{son},1},\nu}^{\Phi,\Upsilon'} \\ \vdots \\ A_{\mu_{\text{son},\ell_\mu},\nu}^{\Phi,\Upsilon'} \end{bmatrix}$;
  else `// (`μ,ν`) is an inadmissible pair of non-leaf clusters and` $j_\mu = j_\nu$
    for $\mu_{\text{son}} \prec \mu$ **and** $\nu_{\text{son}} \prec \nu$ **do**
      compute $A_{\mu_{\text{son}},\nu_{\text{son}}}^{\Phi,\Phi}$:= `recursivelyDetermineMatrixBlock`$(\mu_{\text{son}}, \nu_{\text{son}}, \Phi, \Phi)$;
    return $\left(Q_{\Upsilon}^{\mu}\right)^T \begin{bmatrix} A_{\mu_{\text{son},1},\nu_{\text{son},1}}^{\Phi,\Phi} & \cdots & A_{\mu_{\text{son},1},\nu_{\text{son},\ell_\nu}}^{\Phi,\Phi} \\ \vdots & & \vdots \\ A_{\mu_{\text{son},\ell_\mu},\nu_{\text{son},1}}^{\Phi,\Phi} & \cdots & A_{\mu_{\text{son},\ell_\mu},\nu_{\text{son},\ell_\nu}}^{\Phi,\Phi} \end{bmatrix} Q_{\Upsilon'}^{\nu}$;

Remark 5.7. *The expressions* (5.25) *to* (5.29) *can all be calculated in constant time, as the row and column counts of all involved matrices are bounded by C_Q. This does not mean that a call to* `recursivelyDetermineMatrixBlock` *will always take constant time, however: we did not yet estimate how the impact of* `recursivelyDetermineMatrixBlock` *calling itself recursively. This will be analyzed later.*

Remark 5.8. *As (μ, ν) is not always an element of the block cluster tree (e.g. when the levels of μ and ν differ), the coupling matrix $S^{\mu,\nu}$ might have not been precomputed. It can be computed on the fly in $\mathcal{O}(\#K^2)$ operations. For the $\mathcal{H}^2$-wavelet method, it is in fact advisable* not *to prepare the coupling matrices in advance, as each will be used only a few times while consuming large amounts of memory.*

We shall prove an error estimate:

Lemma 5.9. *Suppose that the submatrices $A^{\Phi,\Phi}_{\mu_{\mathrm{son},i},\mu_{\mathrm{son},i'}}$ from the complete subdivision formula* (5.26) *are each approximated with an error of at most*

$$\left\|A^{\Phi,\Phi}_{\mu_{\mathrm{son},i},\mu_{\mathrm{son},i'}} - \tilde{A}^{\Phi,\Phi}_{\mu_{\mathrm{son},i},\mu_{\mathrm{son},i'}}\right\|_2 < \epsilon, \quad 1 \le i \le \ell_\mu, 1 \le i' \le \ell_\nu.$$

Then, the error of the resulting approximation $\tilde{A}^{\Upsilon,\Upsilon'}_{\mu,\nu}$ is then bounded by $\sqrt{\ell_\mu \ell_\nu}\epsilon$.

Proof. As the multi-scale transformation matrices are submatrices of orthogonal matrices, their spectral norm will be at most 1. This leads to

$$\begin{aligned}
&\left\|A^{\Upsilon,\Upsilon'}_{\mu,\nu} - \tilde{A}^{\Upsilon,\Upsilon'}_{\mu,\nu}\right\|_2 \\
&= \left\| (Q^\mu_\Upsilon)^T \begin{bmatrix} A^{\Phi,\Phi}_{\mu_{\mathrm{son},1},\nu_{\mathrm{son},1}} - \tilde{A}^{\Phi,\Phi}_{\mu_{\mathrm{son},1},\nu_{\mathrm{son},1}} & \cdots & A^{\Phi,\Phi}_{\mu_{\mathrm{son},1},\nu_{\mathrm{son},\ell_\nu}} - \tilde{A}^{\Phi,\Phi}_{\mu_{\mathrm{son},1},\nu_{\mathrm{son},\ell_\nu}} \\ \vdots & & \vdots \\ A^{\Phi,\Phi}_{\mu_{\mathrm{son},\ell_\mu},\nu_{\mathrm{son},1}} - \tilde{A}^{\Phi,\Phi}_{\mu_{\mathrm{son},\ell_\mu},\nu_{\mathrm{son},1}} & \cdots & A^{\Phi,\Phi}_{\mu_{\mathrm{son},\ell_\mu},\nu_{\mathrm{son},\ell_\nu}} - \tilde{A}^{\Phi,\Phi}_{\mu_{\mathrm{son},\ell_\mu},\nu_{\mathrm{son},\ell_\nu}} \end{bmatrix} Q^\nu_{\Upsilon'} \right\|_2 \\
&\le \underbrace{\left\|(Q^\mu_\Upsilon)^T\right\|_2}_{\le 1} \left\| \begin{bmatrix} A^{\Phi,\Phi}_{\mu_{\mathrm{son},1},\nu_{\mathrm{son},1}} - \tilde{A}^{\Phi,\Phi}_{\mu_{\mathrm{son},1},\nu_{\mathrm{son},1}} & \cdots & A^{\Phi,\Phi}_{\mu_{\mathrm{son},1},\nu_{\mathrm{son},\ell_\nu}} - \tilde{A}^{\Phi,\Phi}_{\mu_{\mathrm{son},1},\nu_{\mathrm{son},\ell_\nu}} \\ \vdots & & \vdots \\ A^{\Phi,\Phi}_{\mu_{\mathrm{son},\ell_\mu},\nu_{\mathrm{son},1}} - \tilde{A}^{\Phi,\Phi}_{\mu_{\mathrm{son},\ell_\mu},\nu_{\mathrm{son},1}} & \cdots & A^{\Phi,\Phi}_{\mu_{\mathrm{son},\ell_\mu},\nu_{\mathrm{son},\ell_\nu}} - \tilde{A}^{\Phi,\Phi}_{\mu_{\mathrm{son},\ell_\mu},\nu_{\mathrm{son},\ell_\nu}} \end{bmatrix} \right\|_2 \underbrace{\left\|Q^\nu_{\Upsilon'}\right\|_2}_{\le 1} \\
&\le \sqrt{\sum_{i=1}^{\ell_\mu}\sum_{i'=1}^{\ell_\nu} \left\|A^{\Phi,\Phi}_{\mu_{\mathrm{son},i},\mu_{\mathrm{son},i'}} - \tilde{A}^{\Phi,\Phi}_{\mu_{\mathrm{son},i},\mu_{\mathrm{son},i'}}\right\|_2^2} < \sqrt{\ell_\mu \ell_\nu \epsilon^2} = \sqrt{\ell_\mu \ell_\nu}\epsilon.
\end{aligned}$$

□

The proof of the following corollary is almost identical:

Corollary 5.10 (Error estimate for equations (5.27) and (5.28))**.** *Suppose that the submatrices from the partial subdivision formulae* (5.27) *and* (5.28) *are each approximated with an error of at most ϵ. The error of the resulting approximation $\tilde{A}^{\Upsilon,\Upsilon'}_{\mu,\nu}$ is then bounded by $\sqrt{\ell_\nu}\epsilon$ in the case of* (5.27) *and $\sqrt{\ell_\mu}\epsilon$ for* (5.28)*.*

Combined with Corollary 5.3, this allows us to provide a meaningful bound for the approximation error of an arbitrary matrix block:

Theorem 5.11 (Error estimate for `recursivelyDetermineMatrixBlock`)**.** *Let T be a balanced 2^n-cluster tree with maximum level J and c_{diam} be the constant such that* $\mathrm{diam}(\mu) \sim 2^{-j_\mu}$ *is fulfilled with*

$$c_{\mathrm{diam}} 2^{-j_\mu} \le \mathrm{diam}(\mu) \tag{5.30}$$

for all clusters $\mu \in T$. The error of a matrix block $\tilde{A}^{\Upsilon,\Upsilon'}_{\mu,\nu}, \Upsilon, \Upsilon' \in \{\Phi, \Psi\}$ approximated by means of `recursivelyDetermineMatrixBlock` *is bounded by*

$$\left\|A^{\Upsilon,\Upsilon'}_{\mu,\nu} - \tilde{A}^{\Upsilon,\Upsilon'}_{\mu,\nu}\right\|_2 \le c_0 C_Q 2^{-\frac{j_\mu + j_\nu}{2} n} \left(\frac{\eta 2^J}{c_{\mathrm{diam}}}\right)^{2q+n} \left(\frac{\eta}{\eta + C}\right)^p, \tag{5.31}$$

provided that the error is not dominated by that of the nearfield A^ϕ_{near}.

Proof. In the case of two leaf clusters, the nearfield approximation (which by assumption is sufficiently accurate) is employed.

The next case is that of an admissible pair (μ, ν) where no recursion is necessary. Then, Corollary 5.3 can be applied:

$$\left\|A_{\mu,\nu}^{\Upsilon,\Upsilon'} - \tilde{A}_{\mu,\nu}^{\Upsilon,\Upsilon'}\right\|_2 \leq c_0 C_Q \frac{2^{-(j_\mu+j_\nu)\frac{n}{2}}}{\operatorname{dist}(B_\mu, B_\nu)^{2q+n}} \left(\frac{\eta}{\eta + C}\right)^p. \tag{5.32}$$

From the admissibility condition (4.18) and equation (5.30), we deduce

$$\eta \operatorname{dist}(B_\mu, B_\nu) > \max\{\operatorname{diam}(\mu), \operatorname{diam}(\nu)\} \geq c_{\text{diam}} \max\{2^{-j_\mu}, 2^{-j_\nu}\} = c_{\text{diam}} 2^{-\min\{j_\mu, j_\nu\}} \geq c_{\text{diam}} 2^{-J},$$

which can be inserted into (5.32) for

$$\left\|A_{\mu,\nu}^{\Upsilon,\Upsilon'} - \tilde{A}_{\mu,\nu}^{\Upsilon,\Upsilon'}\right\|_2 < c_0 C_Q 2^{-\frac{j_\mu+j_\nu}{2}n} \left(\frac{\eta 2^J}{c_{\text{diam}}}\right)^{2q+n} \left(\frac{\eta}{\eta + C}\right)^p =: \epsilon_{j_\mu+j_\nu}. \tag{5.33}$$

In the remaining cases, one of the equations (5.26) to (5.28) is used and Lemma 5.9 and Corollary 5.10 (with $\ell_\mu \equiv 2^n$, $\mu \in T$, as we are considering a 2^n-tree) can be employed. This allows us to use induction over the cluster levels:

In the base case of $j_\mu + j_\nu = 2J$, both μ and ν are leaves and the (sufficiently accurate) nearfield matrix will be used. Thus (5.31) holds for all blocks $\tilde{A}_{\mu,\nu}^{\Upsilon,\Upsilon'}$ corresponding to two leaf clusters μ and ν.

Now let $\tilde{j}$ be such that (5.31) holds for all approximated blocks $\tilde{A}_{\mu^\star,\nu^\star}^{\Upsilon^\star,\Upsilon'^\star}$ belonging to any cluster pairs $(\mu^\star, \nu^\star)$ whose cluster levels fulfill $j_{\mu^\star} + j_{\nu^\star} \geq \tilde{j}$. We will show that equation (5.31) then also holds for all approximated blocks $\tilde{A}_{\mu,\nu}^{\Upsilon,\Upsilon'}$ with $j_\mu + j_\nu = \tilde{j} - 1$.

Admissible cluster pairs and pairs of leaf clusters have already been treated above. The other cases are as follows:

- When (5.26) is used, all pairs $(\mu_{\text{son}}, \nu_{\text{son}})$ of son clusters under consideration fulfill $j_{\mu_{\text{son}}} + j_{\nu_{\text{son}}} = j_\mu + 1 + j_\nu + 1 = \tilde{j} + 1$ and thus the induction assumption is applicable:

  $$\left\|A_{\mu_{\text{son}},\nu_{\text{son}}}^{\Phi,\Phi} - \tilde{A}_{\mu_{\text{son}},\nu_{\text{son}}}^{\Phi,\Phi}\right\|_2 < \epsilon_{j_{\mu_{\text{son}}}+j_{\nu_{\text{son}}}} = \epsilon_{(j_\mu+1)+(j_\nu+1)}.$$

  Lemma 5.9 and (5.33) then provide

  $$\left\|A_{\mu,\nu}^{\Upsilon,\Upsilon'} - \tilde{A}_{\mu,\nu}^{\Upsilon,\Upsilon'}\right\|_2 < 2^n \epsilon_{(j_\mu+1)+(j_\nu+1)} = c_0 C_Q 2^{-\frac{j_\mu+j_\nu}{2}n} \left(\frac{\eta 2^J}{c_{\text{diam}}}\right)^{2q+n} \left(\frac{\eta}{\eta + C}\right)^p = \epsilon_{j_\mu+j_\nu}.$$

- For (5.27), we have $j_\mu + j_{\nu_{\text{son}}} = j_\mu + j_\nu + 1 = \tilde{j}$ and thus

  $$\left\|A_{\mu,\nu_{\text{son}}}^{\Upsilon,\Phi'} - \tilde{A}_{\mu,\nu_{\text{son}}}^{\Upsilon,\Phi}\right\|_2 < \epsilon_{j_\mu+j_{\nu_{\text{son}}}} = \epsilon_{j_\mu+(j_\nu+1)},$$

  yielding

  $$\left\|A_{\mu,\nu}^{\Upsilon,\Upsilon'} - \tilde{A}_{\mu,\nu}^{\Upsilon,\Upsilon'}\right\|_2 < 2^{\frac{n}{2}} \epsilon_{j_\mu+(j_\nu+1)} = c_0 C_Q 2^{-\frac{j_\mu+j_\nu}{2}n} \left(\frac{\eta 2^J}{c_{\text{diam}}}\right)^{2q+n} \left(\frac{\eta}{\eta + C}\right)^p = \epsilon_{j_\mu+j_\nu}$$

  with Corollary 5.10 and (5.33).

- Finally, the case of (5.28) is analogous to the previous one.

□

5.4. Setting up the compressed system matrix

We are now able to approximate arbitrary blocks of the multi-scale system matrix. It remains to combine this method with Algorithm 3.2 for an efficient means of setting up the a-priori compressed matrix.

As mentioned in Remark 5.7, calling `recursivelyDetermineMatrixBlock` for an arbitrary block can have non-constant runtime. The naive approach of enumerating all non-negligible blocks by Algorithm 3.2 and then calling `recursivelyDetermineMatrixBlock` for each will therefore take unacceptably long. Fortunately, it is possible to exploit the hierarchical structure of the cluster tree to reuse the matrix blocks for the computation of the corresponding father cluster's blocks, similar to the formulae (5.27) and (5.28).

The general structure of the algorithm is similar to that of Algorithm 3.2, but we now need to examine the column clusters in a particular order: As we reuse the son clusters' matrix blocks, they need to be computed before their father's. Therefore, the column clusters will not simply be iterated over, but instead traversed recursively according to the cluster tree in the function `setupColumn`. For each column cluster, a second recursion `setupRow` similar to `checkCutOffCriterionRecursively` is then performed in order to cover all corresponding non-negligible row clusters. Each call to `setupRow`(μ,ν) will store the corresponding matrix block $\tilde{A}^{\Psi,\Psi}_{\mu,\nu}$ into the compressed system matrix $\tilde{A}^{\Psi}$ and return $\tilde{A}^{\Phi,\Psi}_{\mu,\nu}$ for reuse in the father clusters.

In the recursion for the row clusters ν, we distinguish between leaf and non-leaf column clusters μ, which are treated in the functions `handleRowClusterNonLeaf` and `handleRowClusterLeaf`. Note that these functions have only been introduced for clarity. They could be inlined into `setupRow` and they actually share some variables $(j, j', \tilde{A}^{\Phi,\Psi}_{\mu,\nu}, \tilde{A}^{\Psi,\Psi}_{\mu,\nu})$ with that function.

In `handleRowClusterNonLeaf`, we iterate over μ's son clusters μ_{son} and check if the corresponding cluster pairs belong to non-negligible matrix blocks (i.e. $\operatorname{dist}(B_{\mu_{\text{son}}}, B_\nu) \leq \mathcal{B}_{j_{\mu_{\text{son}}}, j_\nu}$). If they do, the recursion is continued by calling `setupRow`(μ_{son},ν).

If the pair (μ,ν) is admissible, we then employ `recursivelyDetermineMatrixBlock` (which in the case of admissible pairs uses the farfield approximation (5.25)) to directly compute $\tilde{A}^{\Phi,\Psi}_{\mu,\nu}$ and $\tilde{A}^{\Psi,\Psi}_{\mu,\nu}$. Otherwise, the remaining matrices $\tilde{A}^{\Phi,\Psi}_{\mu_{\text{son}},\nu}$ (i.e. those which haven't yet been computed by calling `setupRow`(μ_{son},ν)) are determined via `recursivelyDetermineMatrixBlock`(μ_{son},ν) and used to calculate

$$\begin{bmatrix} \tilde{A}^{\Phi,\Psi}_{\mu,\nu} \\ \tilde{A}^{\Psi,\Psi}_{\mu,\nu} \end{bmatrix} := [Q^\mu_\Phi, Q^\mu_\Psi]^T \begin{bmatrix} \tilde{A}^{\Phi,\Psi}_{\mu_{\text{son},1},\nu} \\ \vdots \\ \tilde{A}^{\Phi,\Psi}_{\mu_{\text{son},\ell_\mu},\nu} \end{bmatrix}.$$

For leaf clusters μ, `setupRow` instead calls `handleRowClusterLeaf`, which in addition to $\tilde{A}^{\Phi,\Psi}_{\mu,\nu}$ and $\tilde{A}^{\Psi,\Psi}_{\mu,\nu}$ (which are needed by `setupRow`) also computes $\tilde{A}^{\Phi,\Phi}_{\mu,\nu}$ and $\tilde{A}^{\Psi,\Phi}_{\mu,\nu}$ for reuse by ν's father clusters. If (μ,ν) is admissible or ν is a leaf cluster, these blocks are again computed by simply calling `recursivelyDetermineMatrixBlock` four times. For inadmissible pairs with ν being not a leaf, the blocks can be taken from

$$\begin{bmatrix} \tilde{A}^{\Phi,\Phi}_{\mu,\nu} & \tilde{A}^{\Phi,\Psi}_{\mu,\nu} \\ \tilde{A}^{\Psi,\Phi}_{\mu,\nu} & \tilde{A}^{\Psi,\Psi}_{\mu,\nu} \end{bmatrix} := \begin{bmatrix} \tilde{A}^{\Phi,\Phi}_{\mu,\nu_{\text{son},1}} & \cdots & \tilde{A}^{\Phi,\Phi}_{\mu,\nu_{\text{son},\ell_\nu}} \\ \tilde{A}^{\Psi,\Phi}_{\mu,\nu_{\text{son},1}} & \cdots & \tilde{A}^{\Psi,\Phi}_{\mu,\nu_{\text{son},\ell_\nu}} \end{bmatrix} [Q^\nu_\Phi, Q^\nu_\Psi].$$

The blocks $\tilde{A}^{\Phi,\Phi}_{\mu,\nu_{\text{son}}}$ and $\tilde{A}^{\Psi,\Phi}_{\mu,\nu_{\text{son}}}$ are provided by either loading them from the store (if they have been stored in a previous call to `handleRowClusterLeaf`) or calling `recursivelyDetermineMatrixBlock` if the clusters satisfy the cut-off condition. This will allow us to later estimate the recursion depth of the call to `recursivelyDetermineMatrixBlock`. The last step consists of storing $\tilde{A}^{\Phi,\Phi}_{\mu,\nu}$ and $\tilde{A}^{\Psi,\Phi}_{\mu,\nu}$ for reuse.

Finally, it remains to compute the matrix entries $\tilde{A}^{\Phi,\Phi}_{\Gamma_N,\Gamma_N}$ and $\tilde{A}^{\Psi,\Phi}_{\mu,\Gamma_N}$, $\mu \in T$ corresponding to scaling functions on the coarsest level. Fortunately, `handleRowClusterLeaf` has already stored the blocks $\tilde{A}^{\Phi,\Phi}_{\mu,\Gamma_N}$ as well as $\tilde{A}^{\Psi,\Phi}_{\mu,\Gamma_N}$ for all leaf clusters $\mu \in L(T)$. They can be used for the computation of the remaining blocks by means of applying a discrete wavelet transform similar to Algorithm 3.1.

These steps are schematically described in Algorithm 5.1 and the subsequent functions.

Algorithm 5.1: Recursive computation of the compressed multi-scale system matrix

Data: Cluster tree T, cut-off parameters a and d', nearfield matrix A^{ϕ}_{near}, multi-scale basis and wavelet transformation matrices V^{μ}_{Φ}, V^{μ}_{Ψ}, Q^{μ}_{Φ}, and Q^{μ}_{Ψ}, $\mu \in T$.

Result: Sparse matrix $\tilde{A}^{\Psi}$ containing the compressed multi-scale system matrix.

begin

- $\tilde{A}^{\Psi} := [0]_{k,k' \in \mathbb{I}^{\Psi}_N}$;
- `setupColumn`(Γ_N);
- store the blocks $\tilde{A}^{\Psi,\Phi}_{\mu,\Gamma_N}, \mu \in L(T)$ (got in `handleRowClusterLeaf`) as part of $\tilde{A}^{\Psi}$;
- apply a partial discrete wavelet transform (cf. Algorithm 3.1) to $\tilde{A}^{\Phi,\Phi}_{\mu,\Gamma_N}$, $\mu \in L(T)$ (also got in `handleRowClusterLeaf`) in order to compute the remaining blocks $\tilde{A}^{\Psi,\Phi}_{\mu,\Gamma_N}$, $\mu \in T \setminus L(T)$ and $\tilde{A}^{\Phi,\Phi}_{\Gamma_N,\Gamma_N}$, then store them as part of $\tilde{A}^{\Psi}$ and erase the blocks $\tilde{A}^{\Phi,\Phi}_{\mu,\Gamma_N}$, $\mu \in L(T)$;

Function setupColumn(ν)

begin

- **for** $\nu_{\text{son}} \prec \nu$ **do**
  - `setupColumn`(ν_{son});
- store $\tilde{A}^{\Phi,\Psi}_{\Gamma_N,\nu} :=$ `setupRow`(Γ_N, ν) as part of $\tilde{A}^{\Psi}$;

Function setupRow(μ, ν)

begin

- **if** μ *is not a leaf* **then**
  - `handleRowClusterNonLeaf`(μ, ν);
- **else**
  - `handleRowClusterLeaf`(μ, ν);
- store $\tilde{A}^{\Psi,\Psi}_{\mu,\nu}$ as part of $\tilde{A}^{\Psi}$;
- **return** $\tilde{A}^{\Phi,\Psi}_{\mu,\nu}$;

Function handleRowClusterNonLeaf(μ, ν)

begin

- **for** $\mu_{\text{son}} \prec \mu$ **do**
  - **if** $\operatorname{dist}(B_{\mu_{\text{son}}}, B_{\nu}) \leq \mathcal{B}_{j_{\mu_{\text{son}}}, j_{\nu}}$ **then**
    - $\tilde{A}^{\Phi,\Psi}_{\mu_{\text{son}},\nu} :=$ `setupRow`(μ_{son}, ν);
- **if** (μ, ν) *is admissible* **then**
  - $\tilde{A}^{\Phi,\Psi}_{\mu,\nu} :=$ `recursivelyDetermineMatrixBlock`(μ, ν, Φ, Ψ);
  - $\tilde{A}^{\Psi,\Psi}_{\mu,\nu} :=$ `recursivelyDetermineMatrixBlock`(μ, ν, Ψ, Ψ);
- **else**
  - **for** $\mu_{\text{son}} \prec \mu$ **do**
    - **if** $\operatorname{dist}(B_{\mu_{\text{son}}}, B_{\nu}) > \mathcal{B}_{j_{\mu_{\text{son}}}, j_{\nu}}$ **then**
      - $\tilde{A}^{\Phi,\Psi}_{\mu_{\text{son}},\nu} :=$ `recursivelyDetermineMatrixBlock`$(\mu_{\text{son}}, \nu, \Phi, \Psi)$;
  - $\begin{bmatrix} \tilde{A}^{\Phi,\Psi}_{\mu,\nu} \\ \tilde{A}^{\Psi,\Psi}_{\mu,\nu} \end{bmatrix} := [Q^{\mu}_{\Phi}, Q^{\mu}_{\Psi}]^T \begin{bmatrix} \tilde{A}^{\Phi,\Psi}_{\mu_{\text{son},1},\nu} \\ \vdots \\ \tilde{A}^{\Phi,\Psi}_{\mu_{\text{son},\ell_{\mu}},\nu} \end{bmatrix}$;

Function handleRowClusterLeaf(μ, ν)

begin

  if (μ,ν) *is admissible or* ν *is a leaf cluster* **then**

    $\tilde{A}^{\Phi,\Phi}_{\mu,\nu} :=$ `recursivelyDetermineMatrixBlock`(μ,ν,Φ,Φ);

    $\tilde{A}^{\Psi,\Phi}_{\mu,\nu} :=$ `recursivelyDetermineMatrixBlock`(μ,ν,Ψ,Φ);

    $\tilde{A}^{\Phi,\Psi}_{\mu,\nu} :=$ `recursivelyDetermineMatrixBlock`(μ,ν,Φ,Ψ);

    $\tilde{A}^{\Psi,\Psi}_{\mu,\nu} :=$ `recursivelyDetermineMatrixBlock`(μ,ν,Ψ,Ψ);

  else

    for $\nu_{\mathrm{son}} \prec \nu$ **do**

      if $\operatorname{dist}(B_\mu, B_{\nu_{\mathrm{son}}}) \leq \mathcal{B}_{j_\mu, j_{\nu_{\mathrm{son}}}}$ **then**

        load stored $\tilde{A}^{\Phi,\Phi}_{\mu,\nu_{\mathrm{son}}}$ and $\tilde{A}^{\Psi,\Phi}_{\mu,\nu_{\mathrm{son}}}$;

      else

        $\tilde{A}^{\Phi,\Phi}_{\mu,\nu_{\mathrm{son}}} :=$ `recursivelyDetermineMatrixBlock`$(\mu,\nu_{\mathrm{son}},\Phi,\Phi)$;

        $\tilde{A}^{\Psi,\Phi}_{\mu,\nu_{\mathrm{son}}} :=$ `recursivelyDetermineMatrixBlock`$(\mu,\nu_{\mathrm{son}},\Psi,\Phi)$;

    $\begin{bmatrix} \tilde{A}^{\Phi,\Phi}_{\mu,\nu} & \tilde{A}^{\Phi,\Psi}_{\mu,\nu} \\ \tilde{A}^{\Psi,\Phi}_{\mu,\nu} & \tilde{A}^{\Psi,\Psi}_{\mu,\nu} \end{bmatrix} := \begin{bmatrix} \tilde{A}^{\Phi,\Phi}_{\mu,\nu_{\mathrm{son},1}} & \dots & \tilde{A}^{\Phi,\Phi}_{\mu,\nu_{\mathrm{son},\ell_\nu}} \\ \tilde{A}^{\Psi,\Phi}_{\mu,\nu_{\mathrm{son},1}} & \dots & \tilde{A}^{\Psi,\Phi}_{\mu,\nu_{\mathrm{son},\ell_\nu}} \end{bmatrix} [Q^\nu_\Phi, Q^\nu_\Psi]$;

  for $\nu_{\mathrm{son}} \prec \nu$ **do**

    if $\operatorname{dist}(B_\mu, B_{\nu_{\mathrm{son}}}) \leq \mathcal{B}_{j_\mu, j_{\nu_{\mathrm{son}}}}$ **then**

      erase stored $\tilde{A}^{\Phi,\Phi}_{\mu,\nu_{\mathrm{son}}}$ and $\tilde{A}^{\Psi,\Phi}_{\mu,\nu_{\mathrm{son}}}$;

  store $\tilde{A}^{\Phi,\Phi}_{\mu,\nu}$ and $\tilde{A}^{\Psi,\Phi}_{\mu,\nu}$;

Remark 5.12. *Algorithm 5.1 is not limited to* symmetric *matrices. To avoid unnecessary computations, a slightly modified version of the algorithm (which takes the symmetry of the single-layer potential operator into account) has been used for obtaining the numerical results in Section 6.*

In order to maintain memory efficiency, the temporary matrix blocks stored by `handleRowClusterLeaf` must not increase the overall storage requirements of the algorithm. This is validated by the following

Lemma 5.13. *The temporary matrix blocks* $\tilde{A}^{\Phi,\Phi}_{\mu,\nu}$ *and* $\tilde{A}^{\Psi,\Phi}_{\mu,\nu}$ *for* $\mu \in L(T)$ *that are stored at the end of* `handleRowClusterLeaf` *require at most* $\mathcal{O}(N\log(N))$ *additional memory.*

Proof. According to the proof of Theorem 3.16, there are $\mathcal{O}(N\log(N))$ cluster pairs that do not satisfy the cut-off condition. As `setupRow`(μ,ν) (and thus `handleRowClusterLeaf`(μ,ν)) only gets called when μ and ν are closer than the cut-off parameter and each call to `handleRowClusterLeaf` only stores *two* matrix blocks, there will be at most $\mathcal{O}(N\log(N))$ temporary blocks in total. Each of these has at most C_Q (i.e. a constant number of) rows and columns, leading to an overall additional memory cost of $\mathcal{O}(N\log(N))$. □

Remark 5.14. *Storing these blocks ensures that* `recursivelyDetermineMatrixBlock` *will only be called for blocks that satisfy the cut-off condition, which in turn is required for the following cost estimates.*

In order to estimate the overall computation time of Algorithm 5.1, we first need to prove that the recursion depth of all calls to `recursivelyDetermineMatrixBlock` made in `handleRowClusterNonLeaf` and `handleRowClusterLeaf` will be bounded by a constant. This implies that the total runtime of each call to `recursivelyDetermineMatrixBlock` is constant as well.

Lemma 5.15 (Recursion depth estimate for calls to `recursivelyDetermineMatrixBlock`)**.** *Let* T *be a balanced* 2^n*-cluster tree and* C_{diam} *be the constant such that* $\operatorname{diam}(B_\mu) \sim 2^{-j_\mu}$ *is fulfilled with*

$$\operatorname{diam}(B_\mu) \leq C_{\mathrm{diam}} 2^{-j_\mu} \tag{5.34}$$

for all clusters $\mu \in T$. Let μ and ν be clusters which correspond to a negligible matrix block, i.e.

$$\operatorname{dist}(B_\mu, B_\nu) > \mathcal{B}_{j,j'}. \tag{5.35}$$

Then the recursion depth of `recursivelyDetermineMatrixBlock`$(\mu, \nu, \Upsilon, \Upsilon')$ *is bounded by a constant*

$$\delta j := \left\lceil \log_2 \left(\frac{C_{\mathrm{diam}}}{\eta a} \right) \right\rceil.$$

In particular, $\max\{j_{\mu^\star} - j_\mu, j_{\nu^\star} - j_\nu\} \leq \delta j$ *holds for all cluster pairs* $(\mu^\star, \nu^\star)$ *that are visited by recursive calls to* `recursivelyDetermineMatrixBlock`*.*

Proof. We begin by proving that all pairs $(\mu^\star, \nu^\star)$ of subclusters $\mu^\star \subset \mu$, $\nu^\star \subset \nu$ which satisfy

$$\min\{j_{\mu^\star}, j_{\nu^\star}\} - \min\{j_\mu, j_\nu\} \geq \log_2 \left(\frac{C_{\mathrm{diam}}}{\eta a} \right)$$

will be admissible, provided that μ and ν fulfill equation (5.35). Therefore, for $(\mu^\star, \nu^\star)$ the farfield approximation can be employed and the recursion ends. It could of course already have ended *earlier*, if $\mu^\star$ and $\nu^\star$ already are leaf clusters, but this does not need to be considered here. To this end, we first insert

$$\operatorname{dist}(B_\mu, B_\nu) > \mathcal{B}_{j_\mu, j_\nu} = a \max \left\{ 2^{-\min\{j_\mu, j_\nu\}}, 2^{\frac{2J(d'-q)-(j_\mu+j_\nu)(d'+\tilde{d})}{2(\tilde{d}+q)}} \right\} \geq a 2^{-\min\{j_\mu, j_\nu\}}.$$

The admissibility condition (4.18) for subclusters $\mu^\star \subset \mu$, $\nu^\star \subset \nu$ reads

$$\max\{\operatorname{diam}(B_{\mu^\star}), \operatorname{diam}(B_{\nu^\star})\} < \eta \operatorname{dist}(B_{\mu^\star}, B_{\nu^\star}).$$

Applying equation (5.34) to the left-hand side yields

$$\max\{\operatorname{diam}(B_{\mu^\star}), \operatorname{diam}(B_{\nu^\star})\} \leq C_{\mathrm{diam}} \max\left\{2^{-j_{\mu^\star}}, 2^{-j_{\nu^\star}}\right\} = C_{\mathrm{diam}} 2^{-\min\{j_{\mu^\star}, j_{\nu^\star}\}}.$$

As

$$\operatorname{dist}(B_{\mu^\star}, B_{\nu^\star}) \geq \operatorname{dist}(B_\mu, B_\nu) > a 2^{-\min\{j_\mu, j_\nu\}}$$

holds, the condition

$$\eta a 2^{-\min\{j_\mu, j_\nu\}} \overset{!}{\geq} C_{\mathrm{diam}} 2^{-\min\{j_{\mu^\star}, j_{\nu^\star}\}}$$

is sufficient for the admissibility of $(\mu^\star, \nu^\star)$. With the definition $\delta j = \min\{j_{\mu^\star}, j_{\nu^\star}\} - \min\{j_\mu, j_\nu\}$, we can rewrite this as

$$2^{\min\{j_{\mu^\star}, j_{\nu^\star}\} - \min\{j_\mu, j_\nu\}} \overset{!}{\geq} \frac{C_{\mathrm{diam}}}{\eta a} \quad \Longleftrightarrow \quad \delta j \overset{!}{\geq} \log_2 \left(\frac{C_{\mathrm{diam}}}{\eta a} \right).$$

Rounding up to the next integer gives

$$\delta j = \left\lceil \log_2 \left(\frac{C_{\mathrm{diam}}}{\eta a} \right) \right\rceil.$$

It remains to show that each recursion step actually corresponds to an increase of the minimum level of the arguments. Otherwise, there could be more than δj recursion steps before the minimum level has actually increased by δj. For this purpose, we show that if `recursivelyDetermineMatrixBlock`$(\mu, \nu, \dots)$ calls `recursivelyDetermineMatrixBlock`$(\overline{\mu}, \overline{\nu}, \dots)$, then $\min\{j_{\overline{\mu}}, j_{\overline{\nu}}\} = \min\{j_\mu, j_\nu\} + 1$ holds. Recall that there are three possibilities which result in a call to `recursivelyDetermineMatrixBlock`$(\overline{\mu}, \overline{\nu}, \dots)$:

1. ν is not a leaf cluster and (μ is a leaf cluster or $j_\mu > j_\nu$).
2. μ is not a leaf cluster and (ν is a leaf cluster or $j_\mu < j_\nu$).

3. Otherwise: neither μ nor ν are leaves and $j_\mu = j_\nu$.

In the first case, `recursivelyDetermineMatrixBlock`$(\mu, \nu_{\text{son}}, \dots)$ is called for a son cluster ν_{son} of ν. As T is a balanced cluster tree, $j_\mu > j_\nu$ also holds in the case of $\mu \in L(T)$. Therefore, due to $j_\mu \geq j_\nu + 1$, we have

$$\min\{j_{\overline{\mu}}, j_{\overline{\nu}}\} = \min\{j_\mu, j_{\nu_{\text{son}}}\} = \min\{j_\mu, j_\nu + 1\} = j_\nu + 1 = \min\{j_\mu, j_\nu\} + 1.$$

The second case is almost identical. In the third case, `recursivelyDetermineMatrixBlock`$(\mu_{\text{son}}, \nu_{\text{son}}, \dots)$ is called for $\mu_{\text{son}} \prec \mu$ and $\nu_{\text{son}} \prec \nu$ with $j_\mu = j_\nu$ and $j_{\mu_{\text{son}}} = j_{\nu_{\text{son}}} = j_\mu + 1$. This leads to

$$\min\{j_{\overline{\mu}}, j_{\overline{\nu}}\} = \min\{j_{\mu_{\text{son}}}, j_{\nu_{\text{son}}}\} = \min\{j_\mu + 1, j_\mu + 1\} = j_\mu + 1 = \min\{j_\mu, j_\nu\} + 1.$$

□

We are now ready to estimate the algorithm's overall runtime and memory cost:

Theorem 5.16 (Overall cost estimate for Algorithm 5.1)**.** *Let T be a balanced 2^n-tree and the nearfield matrix A^ϕ_{near} as well as the multi-scale cluster basis be given. Then, Algorithm 5.1 can then be executed in $\mathcal{O}(N \log(N) \# K^2)$ operations and with storage cost of $\mathcal{O}(N \log(N))$.*

Proof. Note that, similar to Algorithm 3.2 the function `setupRow` is only called for cluster pairs that correspond to non-negligible matrix blocks. There are therefore only $\mathcal{O}(N \log(N))$ calls to `setupRow` (and thus `handleRowClusterNonLeaf` and `handleRowClusterLeaf`) in total (cf. Lemma 3.19). In addition, each of these calls has constant runtime: `setupRow` itself only stores a few constant-size matrix blocks, while `handleRowClusterNonLeaf` and `handleRowClusterLeaf` only call `recursivelyDetermineMatrixBlock` in addition to `setupRow` and only perform a few multiplications of constant-size matrices otherwise. Thanks to the storing of temporary matrix blocks, `recursivelyDetermineMatrixBlock` only gets called in three cases:

1. (μ, ν) is admissible: the $\mathcal{O}(\# K^2)$-effort farfield approximation can be employed.
2. Both μ and ν are leaf clusters: a block $A^\phi_{\mu,\nu}$ of the nearfield matrix can be transformed, taking constant time.
3. The arguments of `recursivelyDetermineMatrixBlock` satisfy the cut-off condition: In this case, Lemma 5.15 can be employed to guarantee a constant recursion depth and $\mathcal{O}(\# K^2)$ overall runtime.

Combining the $\mathcal{O}(\# K^2)$ bound for each call to `recursivelyDetermineMatrixBlock` with the $\mathcal{O}(N \log(N))$-limit for the number of calls, we arrive at a total runtime of $\mathcal{O}(N \log(N) \# K^2)$. The final DWT calls in Algorithm 5.1 (of which there are only C_Q many) do not change this estimate, as they have runtime $\mathcal{O}(N)$ each.

Finally, according to Lemma 5.13, the algorithm only requires $\mathcal{O}(N \log(N))$ temporary memory in addition to the $\mathcal{O}(N \log(N))$ coefficients of $\tilde{A}^\Psi$. □

As the formulae for computing matrix blocks are very similar in `recursivelyDetermineMatrixBlock` and `setupRow`, we can also apply the error estimate (5.31) from Theorem 5.11 to each block computed by Algorithm 5.1. This leads to the following

Theorem 5.17. *Let the conditions of Theorem 5.11 be fulfilled. Then, for the difference between the system matrix A^Ψ_C (compressed according to Theorem 3.16) and its approximated version $\tilde{A}^\Psi$ determined by means of Algorithm 5.1 the following error estimate holds:*

$$\left\| A^\Psi_C - \tilde{A}^\Psi \right\|_2 \lesssim J 2^{Jn} \left(\frac{\eta 2^J}{c_{\text{diam}}} \right)^{2q+n} \left(\frac{\eta}{\eta + C} \right)^p.$$

Proof. As we only compare two versions of the compressed system matrix, it is sufficient to consider the error introduced by approximating non-negligible cluster pairs, of which there are $\sim J 2^{Jn}$ in total (cf. the proof of Theorem 3.16). Multiplying this number with the per-cluster error estimate (5.31) yields the above bound. □

In particular, this estimate is similar to those obtained for the $\mathcal{H}^2$-matrix method [4, 11, 25] and allows us to control the total approximation by increasing p. Choosing $p \sim \log(N)$ allows us to bound the total matrix error with h_N^3, thus guaranteeing the stability and convergence of the complete wavelet Galerkin scheme [14].

6. Numerical Experiments

With an efficient method for setting up the compressed system matrix in the multi-scale basis available, we will now employ the algorithm for actual computations. The algorithm has been implemented in C++, using the Eigen library [10] for linear algebra computations. All tests were run single-threadedly on a compute server which is equipped with one Terabyte of RAM and eight octo-core Intel Xeon X7560 CPUs running at 2.27 GHz. The admissibility parameter was set to $\eta = 1$ for all computations.

We solve the interior Laplace problem with Dirichlet boundary conditions (2.3) by the indirect method for the single layer potential. As the solid harmonics $r^\ell Y_\ell^m(\frac{x}{r})$ are harmonic, they are a suitable choice for the right hand side. In particular, they are eigenfunctions of the single layer potential operator on the sphere with eigenvalues $\lambda_\ell = \frac{1}{2\ell+1}$ [5, 21]. This allows us to directly measure the relative $L_2(\Gamma)$-error

$$E_{L^2} = \frac{\left\|\rho_N - \lambda_\ell^{-1} f\right\|_{L^2(\Gamma)}}{\left\|\lambda_\ell^{-1} f\right\|_{L^2(\Gamma)}} = \frac{1}{\|f\|_{L^2(\Gamma)}} \sqrt{\sum_{i=1}^{N} \int_{\pi_i} \left|\lambda_\ell \rho_i^\phi \phi_i(x) - f(x)\right|^2 \mathrm{d}\sigma_x}$$

of the density on the sphere. For other geometries, only the ℓ^∞-error

$$E_{\ell^\infty} = \max_{x \in X} |\Phi_V(x) - \Phi_{V,N}(x)| = \max_{x \in X} \left| f(x) - \int_\Gamma k_S(x,y) \rho_N(y) \mathrm{d}\sigma_y \right|$$

can be computed for suitable sets X of evaluation points.

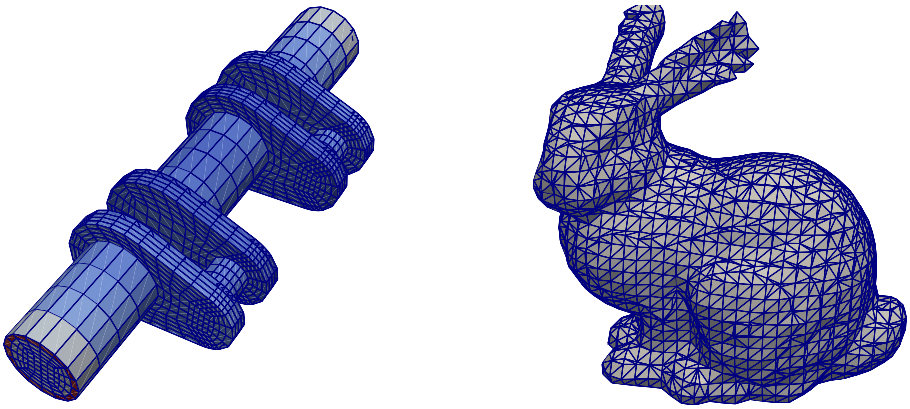

Figure 6.5: A quadrangulation of the crankshaft with two refinements (left) and a triangulation of the Stanford bunny with two pseudo-refinements (right).

Most of the preliminary experiments have been carried out on discretizations of a sphere and a crankshaft (see Figures 2.1 and 6.5). In order to compare the algorithm's performance on quadrangular and triangular meshes, triangular versions of the sphere meshes were obtained by splitting each quadrangle along its diagonal into two triangles.

In order to obtain more meaningful real-world results, the asymptotical behavior of the algorithm was also studied on the so-called "Stanford bunny" [31, 32] (see Figure 6.5). The original bunny model contains five holes at the bottom which were filled by postprocessing with the PolyMender software [18] based on an

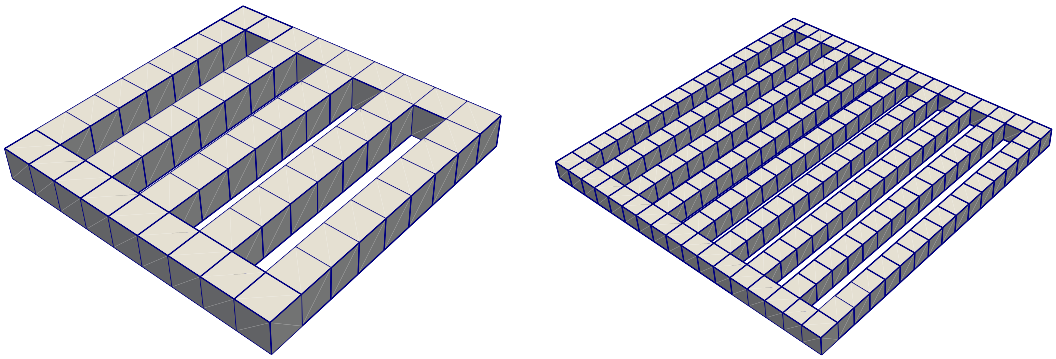

Figure 6.6: Quadrangulations of slit grates with four slits (left) and eight slits (right).

algorithm described in [19]. In addition, slit grates with different slit counts (see Figure 6.6) were generated to study the behavior of the preconditioner.

To increase the accuracy of the boundary discretization, the sphere, crankshaft and slit grate meshes were refined uniformly by splitting the pull-back of each quadrangle (i.e. its representation in the parameter domain) into four new ones. As a result, N is quadrupled and h_N is halved in each step. For the Stanford bunny, the PolyMender package is able to output coarser versions of the original mesh which allow us to introduce a pseudo-refinement level for this geometry as well. Its maximum refinement level is limited by the accuracy of the underlying original polygonal representation, however.

All meshes were shifted and rescaled to fit into a 1-sphere around the origin. This does not impact the overall results, but allows for a better comparison of the cut-off parameter a between different geometries. In addition, it ensures that f attains similar values on all meshes – if a mesh's center was far away from the origin, for example, the values of f (which are proportional to $\|x\|^2$) would become much larger than for a mesh centered at the origin.

In order to fully leverage all the benefits of the $\mathcal{H}^2$-matrix method, we employ the cardinality balanced binary cluster tree (cf. Subsection 3.1) to define admissible cluster pairs in our particular implementation. The Tausch-White wavelets have been defined on 2^n-trees, however. Fortunately, it is possible to combine both trees by using a binary cluster tree in general, but employing *wavelet levels* $\left\lfloor \frac{j_\mu}{n} \right\rfloor$ (rather than regular cluster levels) for the computation of the cut-off parameter.

6.1. Preliminary experiments

In order to obtain any meaningful results, we first need to determine suitable parameters for the algorithm. This subsection is therefore concerned with optimizing the leaf size and the wavelet parameters. In addition, we will make sure that the algorithm delivers comparable results for the quadrangular and triangular discretizations of the same boundary.

6.1.1. Finding the optimal leaf size

The first step is to find a leaf size for which the algorithm's runtime and memory usage are as low as possible. For this purpose, Algorithm 5.1 has been executed for different leaf sizes on the sphere discretized with $N = 6 \cdot 4^M$ quadrangles for $M = 7$ and $M = 9$ with kernel interpolation orders $p = 5$ and $p = 6$, respectively. The wavelet parameters were chosen as $\tilde{d} = 4$, $a = 0.5$ and $d' = 2.8$. Recall that the optimal leaf size for the $\mathcal{H}^2$-matrix method was predicted to be $\#_{\text{leaf,max}} \sim \#K = (p+1)^{n+1}$, while for the wavelets $\#_{\text{leaf,max}} \sim 2m_{\tilde{d}}$ ($= 40$ in the case of $\tilde{d} = 4$) is expected to be optimal (cf. Remark 4.7).

The results of these tests are shown in Figures 6.7 and 6.8. We see that the memory usages of both, the nearfield matrix as well as the compressed wavelet matrix, increase strongly with the leaf size. On the other hand, the nearfield computation time also increases with the leaf size, while the runtime for assembling the compressed system matrix (i.e. executing Algorithm 5.1) is optimal for leaf sizes $\#_{\text{leaf,max}} \sim \#K$. This is

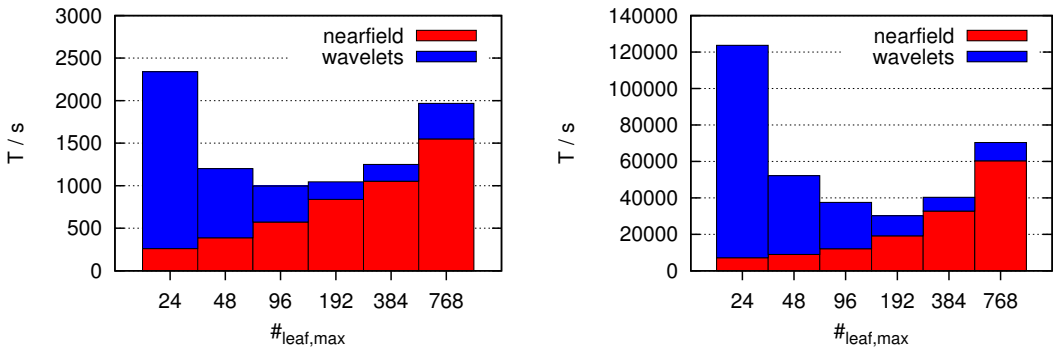

Figure 6.7: Assembly times for different leaf sizes on a sphere with $M = 7$ refinements and $p = 5$ (left) and with $M = 9$ refinements and $p = 6$ (right).

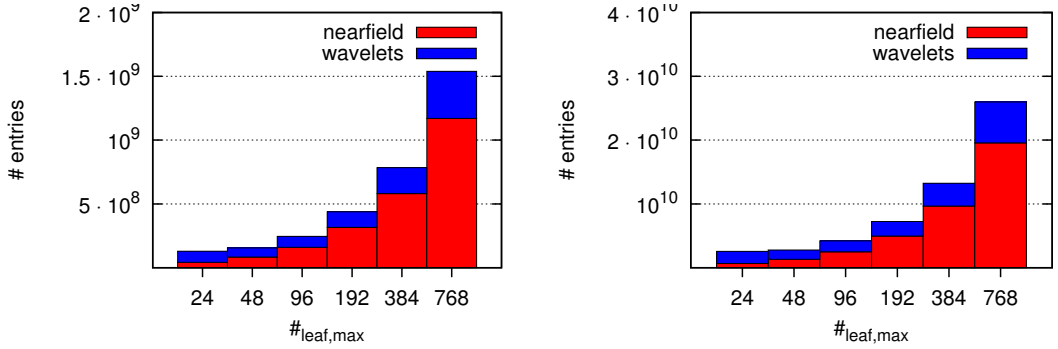

Figure 6.8: Memory usage for different leaf sizes on a sphere with $M = 7$ refinements and $p = 5$ (left) and with $M = 9$ refinements and $p = 6$ (right).

to be expected as the computation of the $(\#K \times \#K)$-coupling matrices has a large impact on the system matrix assembly time. The choice of $\#_{\text{leaf,max}} \sim \#K$ decreases the total number of clusters to $\sim \frac{N}{\#K}$ and therefore significantly reduces the number of coupling matrices that have to be computed. As long as memory is not a concern, it is therefore advisable to choose $\#_{\text{leaf,max}} \sim \#K$ in order to minimize the overall computation time. The algorithm's total memory usage and computation time is then dominated by the nearfield. This also reduces the reduce impact of tweaking the wavelet parameters on the algorithm's overall cost, as we will see in the next section.

6.1.2. Vanishing wavelet moments and cut-off parameters

Now that a sensible leaf size has been found, we still need to determine useful parameters for the wavelet scheme. To this end, computation results for $\tilde{d} = 3$ and $\tilde{d} = 4$ were compared for different values of a on the sphere and the crankshaft. It turns out that the error, runtime and matrix size are nearly independent of a for the sphere. On the other hand, $\tilde{d} = 4$ delivers better results on the crankshaft and optimal convergence is only achieved for $a \geq 1.0$.

The choice of d' has a very low impact on the algorithm's performance. This is plausible as the proof of Theorem 3.16 shows that the d'-dependent part $2^{\frac{2J(d'-q)-(j+j')(d'+\tilde{d})}{2(\tilde{d}+q)}}$ of the cut-off parameter (3.16) only provides $\mathcal{O}(N)$ non-negligible entries to the system matrix, while the $2^{-\min\{j,j'\}}$-part is accountable for $\mathcal{O}(N \log(N))$ entries.

For the remaining experiments, $\tilde{d} = 4$, $a = 0.5$ (except for the crankshaft, where $a = 1.5$ was chosen)

and $d' = 1.5$ have been chosen as simulation parameters. Additional tests have shown that these values are also suitable for the bunny and the slit grate geometries.

6.1.3. Comparison between triangular and quadrangular meshes

As the $\mathcal{H}^2$-wavelet method has been implemented for both *triangular* and *quadrangular* meshes, we are able to directly compare both discretization types. For this purpose, a triangular version of the sphere's original quadrangular discretization was created by splitting each quadrangle into two triangles. Then, the algorithm was run on both meshes with an otherwise identical set of parameters ($p = 6$, $\#_{\text{leaf,max}} = 192$).

The results are listed in Table 6.1. One can see that the memory usage and computation times for the triangular mesh are each about twice as high as on the quadrangular mesh for the same discretization level. This is to be expected as the triangular discretization provides twice as many degrees of freedom as the quadrangular one.

quadrangular mesh							
M	N	$\#_{\text{near}}$	$\#_{\text{wavelet}}$	T_{near}/s	T_{wavelet}/s	E_{ℓ^∞}	E_{L^2}
3	384	$1.11 \cdot 10^5$	$1.04 \cdot 10^5$	0.11	0.02	$1.74 \cdot 10^{-3}$	$1.24 \cdot 10^{-1}$
4	1536	$1.33 \cdot 10^6$	$1.30 \cdot 10^6$	1.53	0.32	$1.06 \cdot 10^{-4}$	$6.07 \cdot 10^{-2}$
5	6144	$1.59 \cdot 10^7$	$6.11 \cdot 10^6$	17.01	4.75	$1.18 \cdot 10^{-5}$	$3.02 \cdot 10^{-2}$
6	24576	$7.55 \cdot 10^7$	$2.72 \cdot 10^7$	108.85	67.58	$1.44 \cdot 10^{-6}$	$1.51 \cdot 10^{-2}$
7	98304	$3.15 \cdot 10^8$	$1.22 \cdot 10^8$	827.85	391.19	$1.84 \cdot 10^{-7}$	$7.54 \cdot 10^{-3}$
8	393216	$1.26 \cdot 10^9$	$5.30 \cdot 10^8$	4388.2	1925.3	$2.23 \cdot 10^{-8}$	$3.77 \cdot 10^{-3}$
9	1572864	$4.97 \cdot 10^9$	$2.25 \cdot 10^9$	19422	9375.3	$3.14 \cdot 10^{-9}$	$1.88 \cdot 10^{-3}$
triangular mesh							
M	N	$\#_{\text{near}}$	$\#_{\text{wavelet}}$	T_{near}/s	T_{wavelet}/s	E_{ℓ^∞}	E_{L^2}
3	768	$3.69 \cdot 10^5$	$3.55 \cdot 10^5$	0.34	0.08	$8.92 \cdot 10^{-4}$	$1.01 \cdot 10^{-1}$
4	3072	$5.01 \cdot 10^6$	$3.23 \cdot 10^6$	4.56	0.86	$3.79 \cdot 10^{-5}$	$4.97 \cdot 10^{-2}$
5	12288	$3.64 \cdot 10^7$	$1.35 \cdot 10^7$	33.36	22.66	$3.90 \cdot 10^{-6}$	$2.47 \cdot 10^{-2}$
6	49152	$1.61 \cdot 10^8$	$5.82 \cdot 10^7$	225.89	161.37	$3.51 \cdot 10^{-7}$	$1.23 \cdot 10^{-2}$
7	196608	$6.56 \cdot 10^8$	$2.53 \cdot 10^8$	1700.2	894.01	$5.48 \cdot 10^{-8}$	$6.17 \cdot 10^{-3}$
8	786432	$2.60 \cdot 10^9$	$1.09 \cdot 10^9$	7280.1	4523.6	$7.41 \cdot 10^{-9}$	$3.08 \cdot 10^{-3}$
9	3145728	$1.02 \cdot 10^{10}$	$4.65 \cdot 10^9$	35327	21236	$1.92 \cdot 10^{-9}$	$1.54 \cdot 10^{-3}$

Table 6.1: Matrix sizes, computation times and errors on the sphere for different refinement levels M with a triangular or a quadrangular discretization.

The ℓ^∞-error is also by a factor of approximately $2^{\frac{3}{2}}$ smaller for the triangular mesh, which corresponds to the two-fold increase in N (and the corresponding decrease in h_N by a factor of about $\sqrt{2}$). Nevertheless, we observe the optimal convergence rate h_N^3 on both types of meshes. Similarly, the L^2-error is also slightly smaller for the triangular discretization compared with the quadrangular mesh on the same level. But again, the expected convergence rate h_N is obtained on both types of meshes.

In total, the algorithm's behavior is very similar for triangular and quadrangular meshes. Therefore, in the following tests a quadrangular discretization – which for the sphere and crankshaft (which contain bilinear quadrangles) is closer to the original geometry than a discretization with flat triangles – will be used where available (i.e. for the sphere, crankshaft and slit grates).

6.2. Asymptotic behavior

It is crucial that the algorithm scales well with the accuracy of the boundary discretization. To verify this, the algorithm has been run on the crankshaft and bunny meshes for different refinement levels. The parameters $p = 4$ was chosen which yields $\#_{\text{leaf,max}} = 71$ for the crankshaft and $\#_{\text{leaf,max}} = 45$ for the bunny.

crankshaft								
M	N	$\#_{\text{near}}$	$\#_{\text{wavelet}}$	% classic	T_{near}/s	T_{wavelet}/s	T_{total}/s	E_{ℓ^∞}
1	568	$1.6 \cdot 10^5$	$1.6 \cdot 10^5$	99.11%	0.13	0.02	0.38	$1.0 \cdot 10^{-2}$
2	2272	$1.8 \cdot 10^6$	$1.7 \cdot 10^6$	68.49%	3.06	0.33	4.44	$4.4 \cdot 10^{-3}$
3	9088	$1.6 \cdot 10^7$	$1.5 \cdot 10^7$	37.93%	44.88	4.13	54.31	$3.5 \cdot 10^{-4}$
4	36352	$8.4 \cdot 10^7$	$9.1 \cdot 10^7$	13.24%	355.29	48.09	429.93	$5.7 \cdot 10^{-5}$
5	145408	$2.8 \cdot 10^8$	$4.6 \cdot 10^8$	3.51%	1704.9	328.36	2173.7	$9.4 \cdot 10^{-6}$
6	581632	$1.0 \cdot 10^9$	$2.1 \cdot 10^9$	0.93%	10106	1711.8	12564	$1.6 \cdot 10^{-6}$
Stanford bunny								
M	N	$\#_{\text{near}}$	$\#_{\text{wavelet}}$	% classic	T_{near}/s	T_{wavelet}/s	T_{total}/s	E_{ℓ^∞}
1	1376	$8.8 \cdot 10^5$	$5.7 \cdot 10^5$	76.23%	1.1	0.23	1.81	$7.0 \cdot 10^{-5}$
2	5572	$5.4 \cdot 10^6$	$3.4 \cdot 10^6$	28.58%	10.39	6.75	20.13	$2.2 \cdot 10^{-5}$
3	22831	$2.3 \cdot 10^7$	$1.8 \cdot 10^7$	7.92%	65.09	52.67	155.43	$5.6 \cdot 10^{-6}$
4	91705	$9.8 \cdot 10^7$	$8.7 \cdot 10^7$	2.21%	388.57	295.29	842.67	$1.2 \cdot 10^{-6}$

Table 6.2: Matrix sizes, compression ratio, computation times and errors for different refinement levels M on the crankshaft and on the bunny.

The results are shown in Table 6.2. Note that the cost for the general preparation and solution steps is not listed separately as they are negligible compared to the cost of assembling and storing the nearfield and compressed system matrices. On the crankshaft and the bunny, the convergence rate of the ℓ^∞-error is somewhat lower than the best possible one which is h_N^{-3} for smooth solutions. Namely, it is about $h_N^{-2.6} \sim 6^{-M}$ and $h_N^{-2} \sim 4^{-M}$, respectively. This is to be expected as these boundaries and the corresponding right-hand side functions are not smooth enough to permit the error estimates given in Proposition 2.1.

The algorithm's memory usage scales as expected. The nearfield's size scales linearly with the number of degrees of freedom (which is proportional to 4^M), while the compressed system matrix asymptotically consists of $\mathcal{O}(N \log(N)) \sim M4^M$ entries (cf. Theorem 3.16). For the bunny, the compressed system matrix actually requires less entries than the nearfield matrix. The additional log-term becomes apparent on the crankshaft, however: There, the compressed system matrix requires more memory than the nearfield for $M \geq 4$. This effect is visible only due to the higher choice of $a = 1.5$ (compared to $a = 0.5$ for the other geometries) which deteriorates the compression.

One can also see from Table 6.2 that the computation times for the nearfield entries is actually more costly than determining the compressed matrix. In particular, one observes a polylog-linear behavior due to the increase of the quadrature order as $|\log(h_N)|$ per quadrature dimension for close panels [25]. The increased computational cost for these close panels is partially offset by the lower quadrature orders (and thus lower cost) for more distant panels, however. The $\mathcal{O}(N \log(N)) \sim M4^M$-asymptotics for the computation of the compressed system matrix can only be seen for larger N as well, as the advantages of the farfield approximation do not become apparent earlier.

Remark 6.1. *On meshes with lower refinement levels, the algorithm's runtime can be reduced by decreasing the interpolation order p which directly affects the $\#K$-part in the runtime estimates [4]. We do not make use of this option, however, as we are only interested in the algorithm's general scaling behavior and doing so would distort its asymptotic complexity.*

6.3. Wavelet preconditioning

We will now verify the viability of the wavelet preconditioning scheme. To this end, the condition numbers of the compressed system matrices and their preconditioned versions

$$\tilde{A}_p^\Psi = \left(\operatorname{diag}\left(\tilde{A}^\Psi\right)\right)^{-\frac{1}{2}} \tilde{A}^\Psi \left(\operatorname{diag}\left(\tilde{A}^\Psi\right)\right)^{-\frac{1}{2}}$$

were computed by employing the Lanczos algorithm to calculate their smallest and largest eigenvalues [6]. In addition to the geometries studied so far, several slit grates with different slit counts were considered [26].

crankshaft					
M	N	cond($\tilde{A}^{\Psi}$)	# iter	cond($\tilde{A}^{\Psi}_p$)	# iter
1	568	177.9	41	8.5	26
2	2272	377.1	66	13.6	30
3	9088	775.5	89	21.3	37
4	36352	1594	115	34.0	45
5	145408	3441	158	53.4	55
6	581632	7337	215	90.7	65
Stanford bunny					
M	N	cond($\tilde{A}^{\Psi}$)	# iter	cond($\tilde{A}^{\Psi}_p$)	# iter
1	1376	9222	287	42.3	56
2	5572	$1.564 \cdot 10^5$	943	1138	127
3	22831	$\sim 2.415 \cdot 10^6$	> 5000	6092	547
4	91705	$\sim 1.805 \cdot 10^7$	> 5000	6086	422

Table 6.3: Degrees of freedom, system matrix condition numbers and CG iteration counts for different refinement levels M on the crankshaft and on the bunny.

First, we listed the condition numbers along with the number of CG iterations necessary to solve the linear system up to a relative error of 10^{-8} in Table 6.3 for the crankshaft and the bunny, respectively. One can see that the condition number of $\tilde{A}^{\Psi}$ approximately doubles with each refinement step as predicted by [25]. In the case of the bunny, the condition numbers increase even more quickly. For $M \geq 3$, neither the CG solver nor the Lanczos iteration converged after 5000 iterations. The effect of the preconditioner is clearly seen for both, the crankshaft and the bunny. The condition numbers as well as the CG-iterations decrease considerably when the wavelet preconditioner is applied. In particular, in case of the bunny, convergence is established again and we arrive at much smaller condition numbers.

# slits	N	cond($\tilde{A}^{\Psi}$)	# iter	cond($\tilde{A}^{\Psi}_p$)	# iter
1	2048	177.1	43	8.30	24
2	4736	270.9	56	16.6	33
4	13184	456.1	72	17.8	34
8	42368	827.8	98	26.0	41
10	63104	1014	109	45.6	51
12	87936	1201	118	39.0	49
16	149888	1575	135	40.8	51
20	228224	1949	147	61.1	58
24	322944	2323	159	62.8	61
28	434048	2697	170	71.5	63
32	561536	3071	180	78.9	68

Table 6.4: Degrees of freedom, system matrix condition numbers and CG iteration counts for several slit grates after $M = 3$ refinements.

Next, we shall consider the slit grates with different slit counts for a fixed refinement level $M = 3$. We observe that the condition numbers of $\tilde{A}^{\Psi}$ for the slit grates with different slit counts are roughly proportional to $\sqrt{N} \sim h_N^{-1}$ (see Table 6.4). Accordingly, the number of CG iterations increases as well. The wavelet preconditioner also works properly for larger slit counts but seems not to be completely robust in the slit

counts (which, indeed, cannot be expected from the theory, see [14]). Nevertheless, the condition numbers are still significantly lower than in the non-preconditioned case.

6.4. Comparison between the $\mathcal{H}^2$-matrix and $\mathcal{H}^2$-wavelet methods

It remains to show that the $\mathcal{H}^2$-wavelet method is actually more efficient than the established $\mathcal{H}^2$-matrix method. Therefore, the calculations from Subsection 6.2 have been repeated using the $\mathcal{H}^2$-matrix method with several appropriate leaf sizes, as can be seen in Table 6.5. The column entitled $\#_{\text{extra}}$ contains the total entry count of either the coupling matrices (in case of the $\mathcal{H}^2$-matrix method) or the compressed system matrix in the multi-scale basis. The preparation time T_{prep} includes general preparations (like computing the nested or multi-scale cluster basis and, for the $\mathcal{H}^2$-matrices, the coupling matrices) as well as the assembly of the nearfield and (in case of the $\mathcal{H}^2$-wavelet method) compressed system matrices. Note that in order to allow the treatment of the bunny geometry with the $\mathcal{H}^2$-matrix method at all, the diagonal of the nearfield matrix was used as a preconditioner, reducing the number of CG iterations from more than 5000 to about 250. This preconditioning scheme did not reduce the number of CG iterations the crankshaft, however.

crankshaft with $M = 5$ refinements						
Parameters	$\#_{\text{near}}$	$\#_{\text{extra}}$	$\#_{\text{total}}$	T_{prep}/s	T_{solve}/s	T_{total}/s
$\mathcal{H}^2$-m., $\#_{\text{leaf}} = 71$	$2.85 \cdot 10^8$	$3.70 \cdot 10^9$	$3.99 \cdot 10^9$	1927.8	2259.0	4186.8
$\mathcal{H}^2$-m., $\#_{\text{leaf}} = 142$	$6.23 \cdot 10^8$	$1.64 \cdot 10^9$	$2.26 \cdot 10^9$	2812.0	939.5	3751.5
$\mathcal{H}^2$-m., $\#_{\text{leaf}} = 284$	$1.28 \cdot 10^9$	$6.41 \cdot 10^8$	$1.92 \cdot 10^9$	4508.5	635.7	5144.2
wavelets, $\#_{\text{leaf}} = 71$	$2.85 \cdot 10^8$	$4.57 \cdot 10^8$	$7.42 \cdot 10^8$	2094.7	79.0	2173.7
Stanford bunny with $M = 4$ refinements						
Parameters	$\#_{\text{near}}$	$\#_{\text{extra}}$	$\#_{\text{total}}$	T_{prep}/s	T_{solve}/s	T_{total}/s
$\mathcal{H}^2$-m., $\#_{\text{leaf}} = 90$	$1.91 \cdot 10^8$	$1.21 \cdot 10^9$	$1.40 \cdot 10^9$	663.3	1216.5	1879.8
$\mathcal{H}^2$-m., $\#_{\text{leaf}} = 180$	$3.59 \cdot 10^8$	$5.74 \cdot 10^8$	$9.33 \cdot 10^8$	980.0	933.0	1913.0
$\mathcal{H}^2$-m., $\#_{\text{leaf}} = 360$	$7.19 \cdot 10^8$	$2.31 \cdot 10^8$	$9.50 \cdot 10^8$	1455.8	1107.9	2563.7
wavelets, $\#_{\text{leaf}} = 45$	$9.81 \cdot 10^7$	$8.74 \cdot 10^7$	$1.86 \cdot 10^8$	712.8	129.9	842.7

Table 6.5: Memory usage and computation times for the $\mathcal{H}^2$-matrix and $\mathcal{H}^2$-wavelet methods on the crankshaft and on the bunny.

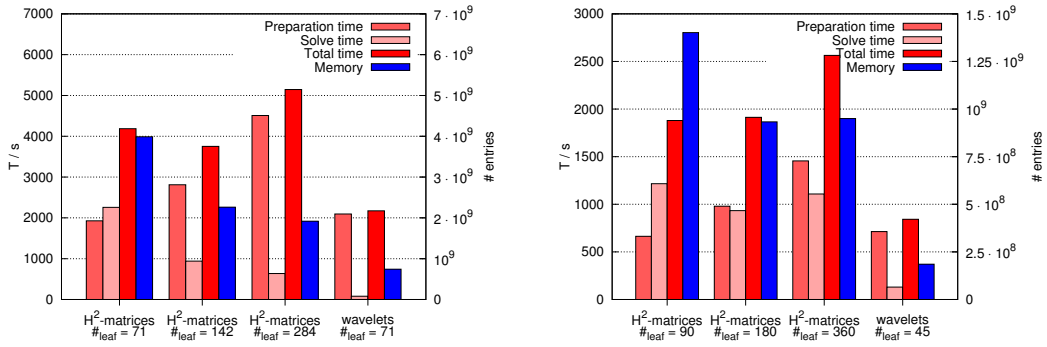

Figure 6.9: Computation times and memory usage for the $\mathcal{H}^2$-matrix and $\mathcal{H}^2$-wavelet methods on the crankshaft with $M = 5$ refinements (left) and on the bunny with $M = 4$ pseudo-refinements (right).

Figures 6.9 illustrate that the $\mathcal{H}^2$-matrices need significantly more memory than the wavelet method. The additional memory is occupied by the coupling matrices (cf. Remark 4.10) – they are used in each matrix-vector multiplication and thus need to be kept in memory. It also becomes apparent that the solution step

is much faster for the wavelet method, as multiplying a (relatively small) block-sparse matrix with a vector is much faster than applying the procedure described in Subsection 4.3. On the crankshaft, this effect is reinforced by a reduction in the number of CG iterations due to the preconditioner employed in the wavelet method. In fact, the additional preparation cost of having to set up the compressed system matrix in the multi-scale basis is already offset after a single solve operation.

We can therefore conclude that the $\mathcal{H}^2$-wavelet method is more time- and memory-efficient than the $\mathcal{H}^2$-matrix method in its current implementation. In order for the $\mathcal{H}^2$-matrices to catch up, additional optimizations (e.g. making use of recompression techniques [3] or a more effective preconditioner for the single-scale matrix [26]) would need to be implemented.

7. Conclusion

In the present paper, we demonstrated the practicability of the $\mathcal{H}^2$-wavelet Galerkin method originally introduced in [14, 20] on actual unstructured meshes in three spatial dimensions. The algorithm's excellent asymptotic behavior for large mesh sizes has been pointed out and its applicability for both triangular and quadrangular meshes has been shown. In addition, its other key advantages became apparent: The $\mathcal{H}^2$-wavelet method has turned out to be both faster and more memory-efficient than the $\mathcal{H}^2$-matrix method. Its very fast matrix-vector multiplication combined with an effective preconditioning scheme result in significantly lower solving times.

Although the algorithm's runtime will probably never match the times presented in [15] (as a parameterization of the geometry allows for several optimizations not applicable in the case of arbitrary geometries), it has been established as a viable tool for computations on geometries where a parametric representation is not available. In future works, the algorithm's behavior could be studied on even larger mesh sizes and different geometries. The Lagrange polynomials employed for the kernel expansion could also be replaced with spherical harmonics. This would reduce the logarithmic terms in the algorithm's complexity estimate and make smaller maximum leaf sizes feasible, thus also reducing the size of the nearfield.

References

[1] M. Bebendorf. Approximation of boundary element matrices. *Numer. Math.*, 86:565–589, 2000.

[2] G. Beylkin, R. Coifman, and V. Rokhlin. The fast wavelet transform and numerical algorithms. *Comm. Pure and Appl. Math.*, 44:141–183, 1991.

[3] S. Börm. Approximation of integral operators by $\mathcal{H}^2$-matrices with adaptive bases. *Computing*, 74(3):249–271, 2005.

[4] S. Börm, M. Löhndorf, and J.M. Melenk. Approximation of integral operators by variable-order interpolation. *Numer. Math.*, 99(4):605–643, 2005.

[5] D. Colton and R. Kress. *Inverse acoustic and electromagnetic scattering theory.* Applied Mathematical Sciences. Springer, New York, 1998.

[6] J.K. Cullum and R.A. Willoughby. *Lanczos algorithms for large symmetric eigenvalue computations. Volume 1, Theory.* Classics in Applied Mathematics. SIAM, Philadelphia, 2002.

[7] W. Dahmen, H. Harbrecht, and R. Schneider. Compression techniques for boundary integral equations. Asymptotically optimal complexity estimates. *SIAM J. Numerical Analysis*, 43(6):2251–2271, 2006.

[8] L. Grasedyck and W. Hackbusch. Construction and arithmetics of $\mathcal{H}$-matrices. *Computing*, 70:295–334, 2003.

[9] L. Greengard and V. Rokhlin. A fast algorithm for particle simulation. *J. Comput. Phys.*, 73:325–348, 1987.

[10] G. Guennebaud, B. Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[11] W. Hackbusch. *Hierarchische Matrizen: Algorithmen und Analysis.* Springer, Berlin, 2009.

[12] W. Hackbusch and S. Börm. $\mathcal{H}^2$-matrix approximation of integral operators by interpolation. *Applied numerical mathematics*, 43(1-2):129–143, 2002.

[13] W. Hackbusch and Z.P. Nowak. On the fast matrix multiplication in the boundary element method by panel clustering. *Numer. Math.*, 54:463–491, 1989.

[14] H. Harbrecht, U. Kähler, and R. Schneider. Wavelet Galerkin BEM on unstructured meshes. *Comput. Vis. Sci.*, 8(3–4):189–199, 2005.

[15] H. Harbrecht and M. Peters. Comparison of fast boundary element methods on parametric surfaces. *Comput. Methods Appl. Mech. Engrg.*, 261–262:39–55, 2013.

[16] H. Harbrecht and R. Schneider. Biorthogonal Wavelet Bases for the Boundary Element Method. *Math. Nach.*, 269–270:167–188, 2004.

[17] H. Harbrecht and R. Schneider. Wavelet Galerkin schemes for boundary integral equations. Implementation and quadrature. *SIAM J. Sci. Comput.*, 27(4):1347–1370, 2005.

[18] T. Ju. PolyMender.
[19] T. Ju. Robust repair of polygonal models. *ACM Trans. Graph*, 23:888–895, 2004.
[20] U. Kähler. *$\mathcal{H}^2$-wavelet Galerkin BEM and its application to the radiosity equation.* PhD thesis, Technische Universität Chemnitz, 2007.
[21] R. Kress. Minimizing the condition number of boundary integral operators in acoustic and electromagnetic scattering. *Quart. J. Mech. Appl. Math.*, 38(2):323–341, 1983.
[22] P. Oswald and M. Hill. Multilevel norms for $H^{-1/2}$. *Computing*, 61(3):235–256, 1998.
[23] T. von Petersdorff and C. Schwab. Wavelet approximation for first kind integral equations on polygons. *Numer. Math.*, 74:479–519, 1996.
[24] S. Rjasanow and O. Steinbach. *The fast solution of boundary integral equations.* Mathematical and analytical techniques with applications to engineering. Springer, New York, 2007.
[25] S.A. Sauter and C. Schwab. *Boundary element methods.* Springer Series in Computational Mathematics. Springer, Berlin-Heidelberg, 2011.
[26] G. Schmidlin. *Fast solution algorithms for integral equations in $\mathbb{R}^3$.* PhD thesis, ETH Zürich, 2003. Nr. 15016, 2002.
[27] R. Schneider. *Multiskalen- und Wavelet-Matrixkompression: analysisbasierte Methoden zur effizienten Lösung großer vollbesetzter Gleichungssysteme.* B.G. Teubner, Stuttgart, 1998.
[28] O. Steinbach. *Numerical approximation methods for elliptic boundary value problems: Finite and boundary elements.* Texts in Applied Mathematics Series. Springer, New York, 2008.
[29] J. Szabados and P. Vértesi. *Interpolation of functions.* World Scientific, Singapore, 1990.
[30] J. Tausch and J. White. Multiscale bases for the sparse representation of boundary integral operators on complex geometry. *SIAM Journal on Scientific Computing*, 24(5):1610–1629 (electronic), 2003.
[31] G. Turk, M. Levoy, et al. The Stanford 3D Scanning Repository.
[32] G. Turk and M. Levoy. Zippered polygon meshes from range images. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '94, pages 311–318. ACM, 1994.

LATEST PREPRINTS

No.	Author: *Title*
2013-01	**H. Harbrecht, M. Peters** *Comparison of Fast Boundary Element Methods on Parametric Surfaces*
2013-02	**V. Bosser, A. Surroca** *Elliptic logarithms, diophantine approximation and the Birch and Swinnerton-Dyer conjecture*
2013-03	**A. Surroca Ortiz** *Unpublished Talk: On some conjectures on the Mordell-Weil and the Tate-Shafarevich groups of an abelian variety*
2013-04	**V. Bosser, A. Surroca** *Upper bound for the height of S-integral points on elliptic curves*
2013-05	**Jérémy Blanc, Jean-Philippe Furter, Pierre-Marie Poloni** *Extension of Automorphisms of Rational Smooth Affine Curves*
2013-06	**Rupert L. Frank, Enno Lenzmann, Luis Silvestre** *Uniqueness of Radial Solutions for the Fractional Laplacian*
2013-07	**Michael Griebel, Helmut Harbrecht** *On the convergence of the combination technique*
2013-08	**Gianluca Crippa, Carlotta Donadello, Laura V. Spinolo** *Initial-Boundary Value Problems for Continuity Equations with BV Coefficients*
2013-09	**Gianluca Crippa, Carlotta Donadello, Laura V. Spinolo** *A Note on the Initial-Boundary Value Problem for Continuity Equations with Rough Coefficients*
2013-10	**Gianluca Crippa** *Ordinary Differential Equations and Singular Integrals*
2013-11	**G. Crippa, M. C. Lopes Filho, E. Miot, H. J. Nussenzveig Lopes** *Flows of Vector Fields with Point Singularities and the Vortex-Wave System*
2013-12	**L. Graff, J. Fender, H. Harbrecht, M. Zimmermann** *Key Parameters in High-Dimensional Systems with Uncertainty*
2013-13	**Jérémy Blanc, Immanuel Stampfli** *Automorphisms of the Plane Preserving a Curve*

LATEST PREPRINTS

No. **Author:** *Title*

2013-14 **Jérémy Blanc, Jung Kyu Canci**
Moduli Spaces of Quadratic Rational Maps with a Marked Periodic Point of Small Order

2013-15 **Marcus J. Grote, Johannes Huber, Drosos Kourounis, Olaf Schenk**
Inexact Interior-Point Method for Pde-Constrained Nonlinear Optimization

2013-16 **Helmut Harbrecht, Florian Loos**
Optimization of Current Carrying Multicables

2013-17 **Daniel Alm, Helmut Harbrecht, Ulf Krämer**
The H^2-Wavelet Method