Max-Planck-Institut
für Mathematik
in den Naturwissenschaften
Leipzig

Hierarchical Matrices

(revised version: June 2006)

by

Steffen Börm, Lars Grasedyck, and Wolfgang Hackbusch

Lecture note no.: 21 2003

Hierarchical Matrices

Steffen Börm Lars Grasedyck Wolfgang Hackbusch

June 6, 2006

Contents

Preface

A major aim of the $\mathcal{H}$-matrix technique is to enable matrix operations of almost linear complexity. Therefore, the notation and importance of linear complexity is discussed in the following.

Linear Complexity

Let $\phi : X_n \to Y_m$ be any function to be evaluated, where n is the number of input data and m is the number of output data. The cost of a computation is at least $\mathcal{O}(N)$ with $N := \max\{n, m\}$, provided that ϕ is a nontrivial mapping (i.e., it is not independent of some of the input data and the output cannot equivalently be described by less than m data). If the cost (or storage) depends linearly on the data size N, we speak about *linear complexity*. The observation from above shows that linear complexity is the optimal one with respect to the order.

Given a class of problems it is essential how the size N of the data behaves. The term "*large scale computation*" expresses that the desired problem size is as large as possible, which of course depends on the actual available computer capacity. Therefore, the size N from above is time-dependent and increases according to the development of the computer technology.

The computer capacity is characterised by two different features: the available storage and the speed of computing. Up to the present time, these quantities increase proportionally, and the same is expected for the future. Under this assumption one can conclude that **large scale computations require linear complexity algorithms**.

For a proof consider a polynomial complexity: Let the arithmetical costs be bounded by CN^σ for some $\sigma \geq 1$. By the definition of a large scale computation, $N = N_t$ should be the maximal size suited for computers at time t. There is a time difference Δt so that at time $t + \Delta t$ the characteristic quantities are doubled: $N_{t+\Delta t} = 2N_t$ and $speed_{t+\Delta t} = 2speed_t$. At time $t + \Delta t$, problems with $N_{t+\Delta t}$ data are to be computed involving $CN^\sigma_{t+\Delta t} = C2^\sigma N_t^\sigma$ operations. Although the number of operations has increased by 2^σ, the improvement concerning the speed leads to an increase of the computer by the factor $2^\sigma/2 = 2^{\sigma-1}$. If $\sigma > 1$ we have the inconvenient situation that the better the computers are, the longer are the running times. Therefore, the only stable situation arises when $\sigma = 1$, which is the case of linear complexity algorithms.

Often, a complexity $\mathcal{O}(N \log^q N)$ for some $q > 0$ appears. We name this "*almost linear complexity*". Since the logarithm increases so slowly, $\mathcal{O}(N \log^q N)$ and $\mathcal{O}(N)$ are quite similar from a practical point of view. If the constant C_1 in $\mathcal{O}(N)$ is much larger than the constant C_2 in $\mathcal{O}(N \log^q N)$, it needs a very big N_t such that $C_2 N_t \log^q N_t \geq C_1 N$, i.e., in the next future $\mathcal{O}(N \log^q N)$ and $\mathcal{O}(N)$ are not distinguishable.

Linear Complexity in Linear Algebra

Linear algebra problems are basic problems which are part of almost all large scale computations, in particular, when they involve partial differential equations. Therefore, it is interesting to check what linear algebra tasks can be performed with linear complexity.

The vector operations (vector sum $x+y$, scalar multiplication λx, scalar product $\langle x, y\rangle$) are obvious candidates

for linear complexity.

However, whenever matrices are involved, the situation becomes worse. The operations Ax, $A+B$, $A*B$, A^{-1}, etc. require $\mathcal{O}(N^2)$ or $\mathcal{O}(N^3)$ operations. The $\mathcal{O}(N^2)$-case can be interpreted as linear complexity, since an $N \times N$-matrix contains N^2 data. However, it is unsatisfactory that one does not know whether a (general) linear system $Ax = b$ (A regular) can be solved by an $\mathcal{O}(N^2)$-algorithm.

Usually, one is working with special subsets of matrices. An ideal family of matrices are the diagonal ones. Obviously, their storage is $\mathcal{O}(N)$ and the standard operations Dx, $D+D'$, $D*D'$, D^{-1} require $\mathcal{O}(N)$ operations. Diagonal matrices allow even to evaluate general functions: $f(D) = \mathrm{diag}\{f(d_i) : 1 \le i \le N\}$.

The class of diagonal matrices might look rather trivial, but is the basis of many FFT applications. If, e.g., C is circulant (i.e., $C_{ij} = C_{i',j'}$ for all $i - j \equiv i' - j' \bmod N$), the Fourier transform by $\mathcal{F}$ leads to a diagonalisation: $\mathcal{F}^{-1}C\mathcal{F} = D$. This allows to inherit all operations mentioned for the diagonal case to the set of circulant matrices. Since the computation of $\mathcal{F}x$ and $\mathcal{F}^{-1}x$ by the fast Fourier transform needs $\mathcal{O}(N \log N)$, these operations are of almost linear complexity.

It is interesting to notice that $\mathcal{F} = \left(\omega^{i+j}\right)_{i,j=1,\ldots,N}$ with $\omega = \exp(2\pi i/N)$ is neither stored nor used (i.e., no entries of $\mathcal{F}$ are called), when $\mathcal{F}x$ is performed by FFT.

Diagonalisation by other matrices than $\mathcal{F}$ is hard to realise. As soon as the transformation T to diagonal form is a full matrix and Tx must be performed in the standard way, the complexity is too high. Furthermore, the possibility of cheap matrix operations is restricted to the subclass of matrices which are simultaneously diagonalised by the transformation.

The class of matrices which is most often used, are the *sparse matrices*, i.e., $\#\{(i,j) : A_{ij} \neq 0\} = \mathcal{O}(N)$. Then, obviously, the storage and the matrix-vector multiplication Ax and the matrix addition (in the same pattern) are of linear complexity. However, already $A*B$ is less sparse, the LU-decomposition $A = LU$ fills the factors L and U, and the inverse A^{-1} is usually dense. Whether $Ax = b$ can be solved in $\mathcal{O}(N)$ operations or not, is not known. The fact that Ax requires only linear complexity is the basis of most of the iterative schemes for solving $Ax = b$.

A subset of the sparse matrices are *band matrices*, at least when the band width ω is $\mathcal{O}(1)$. Here, LU-decomposition costs $\mathcal{O}(\omega^2 N)$ operations, while the band structure of A is inherited by the LU-factors. The disadvantages are similar to those of sparse matrices: $A*B$ has the enlarged band width $\omega_A + \omega_B$ and A^{-1} is dense.

The conclusion is as follows:

- As long as the matrix-vector multiplication $x \mapsto Ax$ is the only desired operation, the sparse matrix format is ideal. This explains the success of the finite element method together with the iterative solution methods.
- Except the diagonal matrices (or diagonal after a certain cheap transformation), there is no class of matrices which allow the standard matrix operations

  $$Ax, \quad A+B, \quad A*B, \quad A^{-1}$$

  in $\mathcal{O}(N)$ operations.

Chapter 1

Introductory Example

In this very first section we introduce the hierarchical matrix structure by a simple one-dimensional model problem. This introduction is along the lines of [40], but here we fix a concrete example that allows us to compute everything explicitly.

1.1 Model Problem

Example 1.1 (One-Dimensional Integral Equation) *We consider the integral equation*

$$\int_0^1 \log|x-y|\,\mathcal{U}(y)\mathrm{d}y = \mathcal{F}(x), \qquad x \in [0,1], \tag{1.1}$$

for a suitable right-hand side $\mathcal{F} : [0,1] \to \mathbb{R}$ *and seek the solution* $\mathcal{U} : [0,1] \to \mathbb{R}$. *The kernel* $g(x,y) = \log|x-y|$ *in* $[0,1]^2$ *has a singularity on the diagonal* $(x = y)$ *and is plotted in Figure 1.1.*

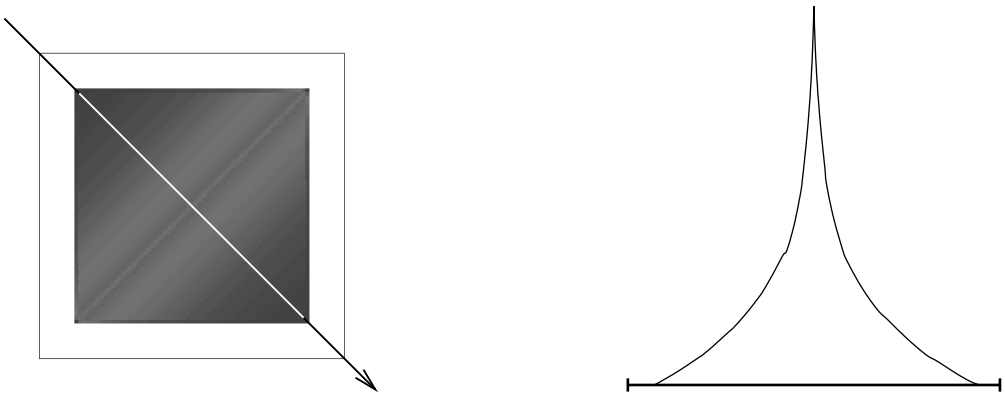


Figure 1.1: The function $\log|x-y|$ in $[0,1] \times [0,1]$ and a cut along the line $x + y = 1$.

A standard discretisation scheme is Galerkin's method where we solve equation (1.1) projected onto the (n-dimensional) space $V_n := \operatorname{span}\{\varphi_0, \ldots, \varphi_{n-1}\}$,

$$\int_0^1 \int_0^1 \varphi_i(x) \log|x-y|\,\mathcal{U}(y)\mathrm{d}y\mathrm{d}x = \int_0^1 \varphi_i(x)\mathcal{F}(x)\mathrm{d}x \qquad 0 \le i < n,$$

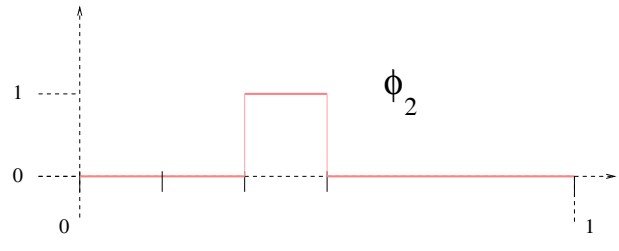
and seek the discrete solution $\mathcal{U}_n$ in the same space V_n, i.e., $\mathcal{U}_n = \sum_{j=0}^{n-1} u_j \varphi_j$ such that the coefficient vector u is the solution of the linear system

$$Gu = f, \quad G_{ij} := \int_0^1 \int_0^1 \varphi_i(x) \log|x-y| \varphi_j(y) \mathrm{d}y \mathrm{d}x, \quad f_i := \int_0^1 \varphi_i(x) \mathcal{F}(x) \mathrm{d}x.$$

In this introductory example we choose piecewise constant basis functions

$$\varphi_i(x) = \begin{cases} 1 & \text{if } \frac{i}{n} \le x \le \frac{i+1}{n} \\ 0 & \text{otherwise} \end{cases} \tag{1.2}$$

on a uniform grid of $[0,1]$:



The matrix G is dense in the sense that all entries are nonzero. Our aim is to approximate G by a matrix $\tilde{G}$ which can be stored in a data-sparse (not necessarily sparse) format. The idea is to replace the kernel $g(x,y) = \log|x-y|$ by a degenerate kernel

$$\tilde{g}(x,y) = \sum_{\nu=0}^{k-1} g_\nu(x) h_\nu(y)$$

such that the integration with respect to the x-variable is separated from the one with respect to the y-variable. However, the kernel function $g(x,y) = \log|x-y|$ cannot be approximated by a degenerate kernel in the whole domain $[0,1] \times [0,1]$ (unless k is rather large). Instead, we construct local approximations for subdomains of $[0,1] \times [0,1]$ where g is smooth (see Figure 1.2).

Figure 1.2: The function $\log|x-y|$ in subdomains of $[0,1] \times [0,1]$.

1.2 Taylor Expansion of the Kernel

Let $\tau := [a,b]$, $\sigma := [c,d]$, $\tau \times \sigma \subset [0,1] \times [0,1]$ be a subdomain with the property $b < c$ such that the intervals are disjoint: $\tau \cap \sigma = \emptyset$. Then the kernel function is nonsingular in $\tau \times \sigma$. Basic calculus reveals

Lemma 1.2 (Derivatives of $\log|x-y|$) *The derivatives of $g(x,y)=\log|x-y|$ for $x\neq y$ and $\nu\in\mathbb{N}$ are*

$$\begin{aligned}\partial_x^\nu g(x,y) &= (-1)^{\nu-1}(\nu-1)!(x-y)^{-\nu}\\ \partial_y^\nu g(x,y) &= -(\nu-1)!(x-y)^{-\nu}.\end{aligned}$$

The Taylor series of $x\mapsto g(x,y)$ in $x_0:=(a+b)/2$ converges in the whole interval τ, and the remainder of the truncated Taylor series can be estimated as follows.

Lemma 1.3 (Taylor series of $\log|x-y|$) *For each $k\in\mathbb{N}$ the function (truncated Taylor series)*

$$\tilde g(x,y):=\sum_{\nu=0}^{k-1}\frac{1}{\nu!}\partial_x^\nu g(x_0,y)(x-x_0)^\nu \tag{1.3}$$

approximates the kernel $g(x,y)=\log|x-y|$ with an error

$$|g(x,y)-\tilde g(x,y)|\leq\frac{|x_0-a|}{|c-b|}\left(1+\frac{|c-b|}{|x_0-a|}\right)^{-k}.$$

Proof: Let $x\in[a,b]$, $a<b$, and $y\in[c,d]$. In the radius of convergence (which we will determine later) the Taylor series of the kernel $g(x,y)$ in x_0 fulfils

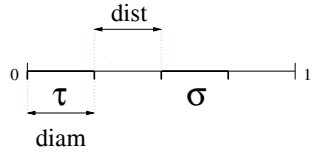
$$g(x,y)=\sum_{\nu=0}^{\infty}\frac{1}{\nu!}\partial_x^\nu g(x_0,y)(x-x_0)^\nu.$$

The remainder $g(x,y)-\tilde g(x,y)=\sum_{\nu=k}^{\infty}\frac{1}{\nu!}\partial_x^\nu g(x_0,y)(x-x_0)^\nu$ can be estimated by

$$\begin{aligned}\Big|\sum_{\nu=k}^{\infty}\frac{1}{\nu!}\partial_x^\nu g(x_0,y)(x-x_0)^\nu\Big| &= \Big|\sum_{\nu=k}^{\infty}(-1)^{\nu-1}\frac{(\nu-1)!}{\nu!}\left(\frac{x-x_0}{x_0-y}\right)^\nu\Big|\\ &\leq \sum_{\nu=k}^{\infty}\Big|\frac{x-x_0}{x_0-y}\Big|^\nu\\ &\leq \sum_{\nu=k}^{\infty}\left(\frac{|x_0-a|}{|x_0-a|+|c-b|}\right)^\nu\\ &= \left(1+\frac{|x_0-a|}{|c-b|}\right)\left(1+\frac{|c-b|}{|x_0-a|}\right)^{-k}.\end{aligned}$$

Since $1+\frac{|c-b|}{|x_0-a|}>1$, the radius of convergence covers the whole interval $[a,b]$. ∎

If $c\to b$ then the estimate for the remainder tends to infinity and the approximation can be arbitrarily bad. However, if we replace the condition $b<c$, i.e., the disjointness of the intervals, by the stronger **admissibility condition**



$$\operatorname{diam}(\tau)\leq\operatorname{dist}(\tau,\sigma) \tag{1.4}$$

then the approximation error can be estimated by

$$|g(x,y)-\tilde g(x,y)|\leq\frac{3}{2}(1+\frac{2}{1})^{-k}\leq\frac{3}{2}3^{-k}. \tag{1.5}$$

This means we get a uniform bound for the approximation error independently of the intervals as long as the admissibility condition is fulfilled. The error decays exponentially with respect to the order k.

1.3 Low Rank Approximation of Matrix Blocks

The index set $\mathcal{I} := \{0, \ldots, n-1\}$ contains the indices of the basis functions φ_i used in the Galerkin discretisation. We fix two subsets t and s of the index set $\mathcal{I}$ and define the corresponding domains as the union of the supports of the basis functions:

$$\tau := \bigcup_{i \in t} \operatorname{supp} \varphi_i, \qquad \sigma := \bigcup_{i \in s} \operatorname{supp} \varphi_i.$$

If $\tau \times \sigma$ is admissible with respect to (1.4), then we can approximate the kernel g in this subdomain by the truncated Taylor series $\tilde{g}$ from (1.3) and replace the matrix entries

$$G_{ij} = \int_0^1 \int_0^1 \varphi_i(x) g(x,y) \varphi_j(y) \mathrm{d}y \mathrm{d}x$$

by use of the degenerate kernel $\tilde{g} = \sum_{\nu=0}^{k-1} g_\nu(x) h_\nu(y)$ for the indices $(i,j) \in t \times s$:

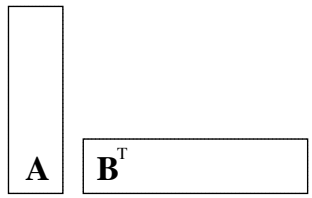
$$\tilde{G}_{ij} := \int_0^1 \int_0^1 \varphi_i(x) \tilde{g}(x,y) \varphi_j(y) \mathrm{d}y \mathrm{d}x.$$

The benefit is twofold. First, the double integral is separated in two single integrals:

$$\begin{aligned} \tilde{G}_{ij} &= \int_0^1 \int_0^1 \varphi_i(x) \sum_{\nu=0}^{k-1} g_\nu(x) h_\nu(y) \varphi_j(y) \mathrm{d}y \mathrm{d}x \\ &= \sum_{\nu=0}^{k-1} \left(\int_0^1 \varphi_i(x) g_\nu(x) \mathrm{d}x \right) \left(\int_0^1 \varphi_j(y) h_\nu(y) \mathrm{d}y \right). \end{aligned}$$

Second, the submatrix $G|_{t \times s}$ can be represented in factorised form

$$G|_{t \times s} = AB^T, \qquad A \in \mathbb{R}^{t \times \{0,\ldots,k-1\}}, \quad B \in \mathbb{R}^{s \times \{0,\ldots,k-1\}},$$



where the entries of the factors A and B are

$$A_{i\nu} := \int_0^1 \varphi_i(x) g_\nu(x) \mathrm{d}x, \qquad B_{j\nu} := \int_0^1 \varphi_j(y) h_\nu(y) \mathrm{d}y.$$

The rank of the matrix AB^T is at most k independently of the cardinality of t and s. The approximation error of the matrix block is estimated in the next lemma.

Definition 1.4 (Admissible Blocks) *A block $t \times s \subset \mathcal{I} \times \mathcal{I}$ of indices is called admissible if the corresponding domain $\tau \times \sigma$ with $\tau := \cup_{i \in t} \operatorname{supp} \varphi_i$, $\sigma := \cup_{i \in s} \operatorname{supp} \varphi_i$ is admissible in the sense of (1.4).*

Lemma 1.5 (Local Matrix Approximation Error) *The elementwise error for the matrix entries G_{ij} approximated by the degenerate kernel $\tilde{g}$ in the admissible block $t \times s$ (and g in the other blocks) is bounded by*

$$|G_{ij} - \tilde{G}_{ij}| \leq \frac{3}{2} n^{-2} 3^{-k}.$$

Proof:

$$|G_{ij} - \tilde{G}_{ij}| = \left| \int_0^1 \int_0^1 \varphi_i(x) (g(x,y) - \tilde{g}(x,y)) \varphi_j(y) \mathrm{d}y \mathrm{d}x \right|$$

$$\begin{aligned} &\overset{(1.5)}{\leq} && \int_0^1 \int_0^1 |\varphi_i(x)| 3^{-k} |\varphi_j(y)| \mathrm{d}y \mathrm{d}x \\ &= && \frac{3}{2} 3^{-k} \int_0^1 \varphi_i(x) \mathrm{d}x \int_0^1 \varphi_j(y) \mathrm{d}y \\ &= && \frac{3}{2} n^{-2} 3^{-k}. \end{aligned}$$

■

Let us assume that we have partitioned the index set $\mathcal{I} \times \mathcal{I}$ for the matrix G into admissible blocks, where the low rank approximation is applicable, and inadmissible blocks, where we use the matrix entries from G (in the next two subsections we will present a constructive method for the partitioning):

$$\mathcal{I} \times \mathcal{I} = \dot{\bigcup}_{t \times s \in \mathcal{P}} t \times s.$$

Then the global approximation error is estimated in the Frobenius norm $\|M\|_F^2 := \sum M_{ij}^2$:

Lemma 1.6 (Matrix Approximation Error) *The approximation error $\|G - \tilde{G}\|_F$ in the Frobenius norm for the matrix $\tilde{G}$ built by the degenerate kernel $\tilde{g}$ in the admissible blocks $t \times s \in \mathcal{P}$ and by g in the inadmissible blocks is bounded by*

$$\|G - \tilde{G}\|_F \leq \frac{3}{2} n^{-1} 3^{-k}.$$

Proof: Apply Lemma 1.5. ■

The question remains, how we want to partition the product index set $\mathcal{I} \times \mathcal{I}$ into admissible and inadmissible blocks. A trivial partition would be $\mathcal{P} := \{(i, j) \mid i \in \mathcal{I}, j \in \mathcal{I}\}$ where only 1×1 blocks of rank 1 appear. In this case the matrix $\tilde{G}$ is identical to G, but we do not exploit the option to approximate the matrix in large subblocks by matrices of low rank.

The number of possible partitions of $\mathcal{I} \times \mathcal{I}$ is rather large (even the number of partitions of $\mathcal{I}$ is so). In subsequent chapters we will present general strategies for the construction of suitable partitions, but here we only give an exemplary construction.

1.4 Cluster Tree $T_{\mathcal{I}}$

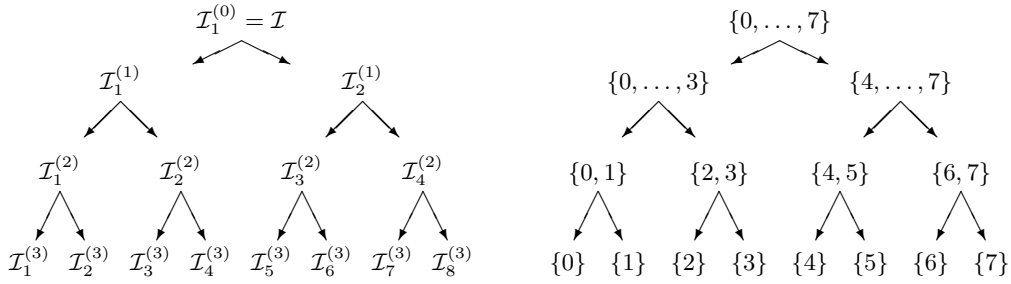


Figure 1.3: The cluster tree $T_{\mathcal{I}}$ for $p = 3$, on the left abstract and on the right concrete.

The candidates $t, s \subset \mathcal{I}$ for the construction of the partition of $\mathcal{I} \times \mathcal{I}$ will be stored in a so-called *cluster tree* $T_{\mathcal{I}}$. The root of the tree $T_{\mathcal{I}}$ is the index set $\mathcal{I}_1^{(0)} := \{0, \ldots, n-1\}$. For ease of presentation we assume the number n of basis functions to be a power of two:

$$n = 2^p.$$

The two successors of $\mathcal{I}_1^{(0)}$ are $\mathcal{I}_1^{(1)} := \{0, \ldots, \frac{n}{2} - 1\}$ and $\mathcal{I}_2^{(1)} := \{\frac{n}{2}, \ldots, n-1\}$.

The two successors of $\mathcal{I}_1^{(1)}$ are $\mathcal{I}_1^{(2)} := \{0, \ldots, \frac{n}{4} - 1\}$ and $\mathcal{I}_2^{(2)} := \{\frac{n}{4}, \ldots, \frac{n}{2} - 1\}$.

The two successors of $\mathcal{I}_2^{(1)}$ are $\mathcal{I}_3^{(2)} := \{\frac{n}{2}, \ldots, 3\frac{n}{4} - 1\}$ and $\mathcal{I}_4^{(2)} := \{3\frac{n}{4}, \ldots, n-1\}$.

Each subsequent node t **with more than** $n_{\min}$ **indices** has exactly two successors: the first contains the first half of its indices, the second one the second half. Nodes with not more than $n_{\min}$ indices are leaves. The parameter $n_{\min}$ controls the depth of the tree. For $n_{\min} = 1$ we get the maximal depth. However, for practical purposes (e.g., if the rank k is larger) we might want to set $n_{\min} = 2k$ or $n_{\min} = 16$.

Remark 1.7 (Properties of $T_{\mathcal{I}}$) *For $n_{\min} = 1$ the tree $T_{\mathcal{I}}$ is a binary tree of depth p (see Figure 1.3). It contains subsets of the index set $\mathcal{I}$ of different size. The first level consists of the root $\mathcal{I} = \{0, \ldots, n-1\}$ with n indices, the second level contains two nodes with $n/2$ indices each and so forth, i.e., the tree is* cardinality *balanced. The number of nodes in the cardinality balanced binary tree $T_{\mathcal{I}}$ (for $n_{\min} = 1$) is $\#T_{\mathcal{I}} = 2n - 1$.*

1.5 Block Cluster Tree $T_{\mathcal{I}\times\mathcal{I}}$

The number of possible blocks $t \times s$ with nodes t, s from the tree $T_{\mathcal{I}}$ is $(\#T_{\mathcal{I}})^2 = (2n-1)^2 = \mathcal{O}(n^2)$. This implies that we cannot test *all* possible combinations (our aim is to reduce the quadratic cost for the assembly of the matrix).

One possible method is to test blocks level by level starting with the root $\mathcal{I}$ of the tree $T_{\mathcal{I}}$ and descending in the tree. The tested blocks are stored in a so-called *block cluster tree* $T_{\mathcal{I}\times\mathcal{I}}$ whose leaves form a partition of the index set $\mathcal{I}\times\mathcal{I}$. The algorithm is given as follows and called with parameters BuildBlockClusterTree($\mathcal{I}$,$\mathcal{I}$).

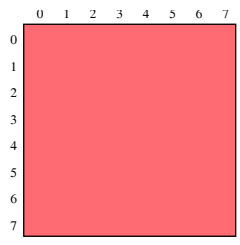
Algorithm 1 Construction of the block cluster tree $T_{\mathcal{I}\times\mathcal{I}}$

```
procedure BuildBlockClusterTree(cluster t, s)
if (t, s) is admissible then
  S(t × s) := ∅
else
  S(t × s) := {t' × s' | t' ∈ S(t) and s' ∈ S(s)}
  for t' ∈ S(t) and s' ∈ S(s) do
    BuildBlockClusterTree(t', s')
  end for
end if
```

The tree $T_{\mathcal{I}\times\mathcal{I}}$ is a quadtree, but there are leaves on different levels of the tree which is not the case for the binary tree $T_{\mathcal{I}}$.

Example 1.8 (Block cluster tree, $p = 3$) *We consider the example tree from Figure 1.3. The root of the tree is*

$$\{0, \ldots, 7\} \times \{0, \ldots, 7\}$$

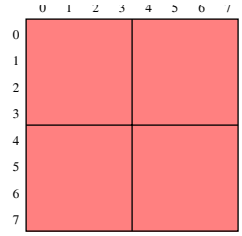


which is not admissible because the corresponding domain to the index set $\{0, \ldots, 7\}$ is the interval $[0, 1]$ and

$$\operatorname{diam}([0,1]) = 1 \not\leq 0 = \operatorname{dist}([0,1],[0,1]).$$

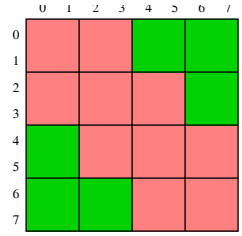
The four successors of the root in the tree $T_{\mathcal{I}\times\mathcal{I}}$ are

$$\{0,1,2,3\}\times\{0,1,2,3\}, \quad \{0,1,2,3\}\times\{4,5,6,7\},$$
$$\{4,5,6,7\}\times\{0,1,2,3\}, \quad \{4,5,6,7\}\times\{4,5,6,7\}.$$



Again, none of these is admissible, and they are further subdivided into

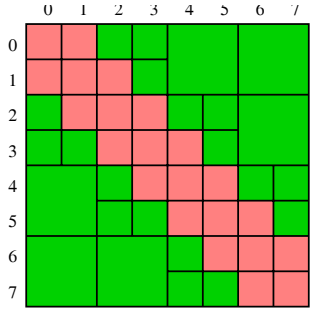
$$\begin{array}{llll}
\{0,1\}\times\{0,1\}, & \{0,1\}\times\{2,3\}, & \{0,1\}\times\{4,5\}, & \{0,1\}\times\{6,7\},\\
\{2,3\}\times\{0,1\}, & \{2,3\}\times\{2,3\}, & \{2,3\}\times\{4,5\}, & \{2,3\}\times\{6,7\},\\
\{4,5\}\times\{0,1\}, & \{4,5\}\times\{2,3\}, & \{4,5\}\times\{4,5\}, & \{4,5\}\times\{6,7\},\\
\{6,7\}\times\{0,1\}, & \{6,7\}\times\{2,3\}, & \{6,7\}\times\{4,5\}, & \{6,7\}\times\{6,7\}.
\end{array}$$



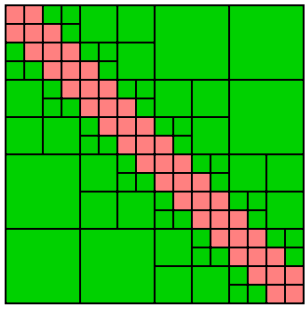
Now some of the nodes are admissible, e.g., the node $\{0,1\}\times\{4,5\}$ because the corresponding domain is $[0,\frac{1}{4}]\times[\frac{1}{2},\frac{3}{4}]$:

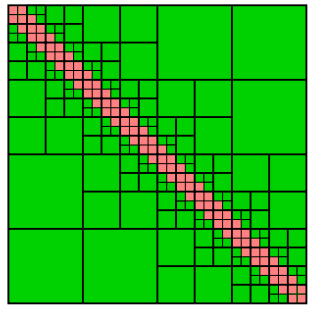
$$\operatorname{diam}\left(\left[0,\frac{1}{4}\right]\right) = \frac{1}{4} = \operatorname{dist}\left(\left[0,\frac{1}{4}\right],\left[\frac{1}{2},\frac{3}{4}\right]\right).$$

The nodes on the diagonal are not admissible (the distance of the corresponding domain to itself is zero) and also some nodes off the diagonal, e.g., $\{0,1\}\times\{2,3\}$, are not admissible. The successors of these nodes are the singletons $\{(i,j)\}$ for indices i,j. The final structure of the partition looks as follows:



For $p=4$ and $p=5$ the structure of the partition is similar:





1.6 Assembly, Storage and Matrix-Vector Multiplication

The product index set $\mathcal{I} \times \mathcal{I}$ resolves into admissible and inadmissible leaves of the tree $T_{\mathcal{I}\times\mathcal{I}}$. The assembly, storage and matrix-vector multiplication differs for the corresponding two classes of submatrices.

1.6.1 Inadmissible Leaves

In the **inadmissible** (but small !) blocks $t \times s \subset \mathcal{I} \times \mathcal{I}$ we compute the entries (i, j) as usual:

$$\begin{aligned} \tilde{G}_{ij} &:= \int_0^1 \int_0^1 \varphi_i(x) \log|x-y| \varphi_j(y) \mathrm{d}y \mathrm{d}x \\ &= \int_{i/n}^{(i+1)/n} \int_{j/n}^{(j+1)/n} \log|x-y| \mathrm{d}y \mathrm{d}x. \end{aligned}$$

Definition 1.9 (`fullmatrix` Representation) *An $n \times m$ matrix M is said to be stored in* `fullmatrix` *representation if the entries M_{ij} are stored as real numbers (*`double`*) in an array of length nm in the order*

$$M_{11}, \ldots, M_{n1}, M_{12}, \ldots, M_{n2}, \quad \ldots \quad, M_{1m}, \ldots, M_{nm} \qquad \textit{(column-wise).}$$

Implementation 1.10 (`fullmatrix`) *The* `fullmatrix` *representation in the* C *programming language might look as follows:*

```
typedef struct _fullmatrix fullmatrix;
typedef fullmatrix *pfullmatrix;

struct _fullmatrix {
  int rows;
  int cols;
  double* e;
};
```

The array `e` *has to be allocated and deallocated dynamically in the constructor and destructor:*

```
pfullmatrix
new_fullmatrix(int rows, int cols){
  pfullmatrix f = (pfullmatrix) malloc(sizeof(fullmatrix));
  f->rows = rows;
  f->cols = cols;
  f->e = (double*) malloc(rows*cols*sizeof(double));
  for(i=0; i<rows*cols; i++) f->e[i] = 0.0;
  return f;
}

void
del_fullmatrix(pfullmatrix f){
  if(f->e) free(f->e);
  free(f);
  f = 0x0;
}
```

The entry M_{ij} is stored at the position `f->e[i + j * f->rows]`. The ordering of the matrix entries in the fullmatrix representation is the same ordering that is used in standard linear algebra packages (BLAS, LAPACK, MATLAB, etc.). Therefore, procedures from these libraries can be called without complications, e.g., the matrix-vector multiplication can be performed by calling the standard BLAS subroutine `dgemv`.

1.6.2 Admissible Leaves

In the **admissible** blocks $t \times s \subset \mathcal{I} \times \mathcal{I}$ with corresponding domains $[a,b] \times [c,d]$ and $x_0 := (a+b)/2$ we compute the submatrix in factorised form

$$\begin{aligned} \tilde{G}|_{t\times s} &:= AB^T, \\ A_{i\nu} &:= \int_{i/n}^{(i+1)/n} (x - x_0)^\nu \mathrm{d}x, \\ B_{j\nu} &:= \begin{cases} (-1)^{\nu+1}\nu^{-1} \int_{j/n}^{(j+1)/n} (x_0 - y)^{-\nu} \mathrm{d}y & \text{if } \nu > 0 \\ \int_{j/n}^{(j+1)/n} \log|x_0 - y| \mathrm{d}y & \text{if } \nu = 0. \end{cases} \end{aligned}$$

Definition 1.11 ($\mathcal{R}(k)$-Matrices) *Let $n, m \in \mathbb{N}, k \in \mathbb{N}_0$. We define the set of $n \times m$ matrices of rank at most k by*

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

A suitable representation for the submatrix $\tilde{G}|_{t\times s}$ is the `rkmatrix` format defined next.

Definition 1.12 (`rkmatrix` Representation) *An $n \times m$ matrix M of rank at most k is said to be stored in `rkmatrix` representation if it is stored in factorised form $M = AB^T$ where the two matrices $A \in \mathbb{R}^{n\times k}$ and $B \in \mathbb{R}^{m\times k}$ are both stored as an array (column-wise).*

Implementation 1.13 (`rkmatrix`) *The `rkmatrix` representation is implemented in the* C *programming language as follows:*

```
typedef struct _rkmatrix rkmatrix;
typedef rkmatrix *prkmatrix;

struct _rkmatrix {
  int k;
  int kt;
  int rows;
  int cols;
  double* a;
  double* b;
};
```

The arrays `a` *and* `b` *have to be allocated and deallocated dynamically in the constructor and destructor:*

```
prkmatrix
new_rkmatrix(int k, int rows, int cols){
  int i;
  prkmatrix r = (prkmatrix) malloc(sizeof(rkmatrix));
  r->k = k;
  r->kt = 0;
  r->rows = rows;
  r->cols = cols;
  r->a = 0x0;
  r->b = 0x0;
  if(k>0){
    r->a = (double*) malloc(k*rows*sizeof(double));
```

```
    for(i=0; i<rows*k; i++) r->a[i] = 0.0;
    r->b = (double*) malloc(k*cols*sizeof(double));
    for(i=0; i<cols*k; i++) r->b[i] = 0.0;
  }
  return r;
}

void
del_rkmatrix(prkmatrix r){
  free(r->a);
  free(r->b);
  free(r);
}
```

The implementation of the `fullmatrix` *representation for the factors A and B in the implementation of the* `rkmatrix` *representation differs from the one in Implementation 1.10: the information about the size of the two factors is stored in the* `rkmatrix` *structure and the two matrices A and B are stored as two arrays. The integer* `k` *in the* `rkmatrix` *structure is an upper bound for the allowed rank while* `kt` *denotes the actual (temporarily) used rank. Often* `k` *equals* `kt`*, but in general there holds* `kt` $\leq$ `k`*.*

1.6.3 Hierarchical Matrix Representation

Definition 1.14 ($\mathcal{H}$-matrix) *Let $T_{\mathcal{I}\times\mathcal{I}}$ be a block cluster tree for the index set $\mathcal{I}$. We define the set of $\mathcal{H}$-matrices as*

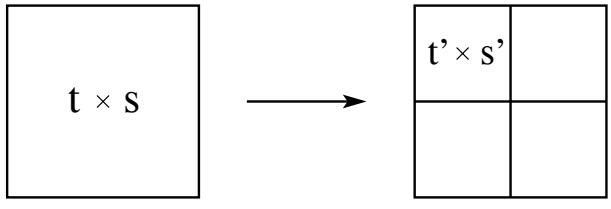
$$\mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, k) := \left\{ M \in \mathbb{R}^{\mathcal{I}\times\mathcal{I}} \;\middle|\; \operatorname{rank}(M|_{t\times s}) \leq k \quad \text{for all admissible leaves } t\times s \text{ of } T_{\mathcal{I}\times\mathcal{I}} \right\}.$$

Definition 1.15 ($\mathcal{H}$-matrix Representation) *Let $T_{\mathcal{I}\times\mathcal{I}}$ be a block cluster tree for the index set $\mathcal{I}$. A matrix $M \in \mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, k)$ is said to be stored in $\mathcal{H}$-matrix representation if the submatrices corresponding to inadmissible leaves are stored in* `fullmatrix` *representation and those corresponding to admissible leaves are stored in* `rkmatrix` *representation.*

One possible implementation for a matrix in $\mathcal{H}$-matrix representation would be to store the admissible and inadmissible matrix blocks in a list. The assembly and matrix-vector multiplication can be done for each block separately. However, we choose a different implementation that is guided by the block tree $T_{\mathcal{I}\times\mathcal{I}}$ (not only the leaves) and stores the matrix in a more "structured" way.

Each block $t \times s$ in the tree $T_{\mathcal{I}\times\mathcal{I}}$ can be

- a leaf - then the corresponding matrix block is represented by a `fullmatrix` or `rkmatrix`;
- not a leaf - then the block $t \times s$ is decomposed into its sons $t' \times s'$ with $t' \in S(t)$ and $s' \in S(s)$. This means that the matrix corresponding to the block $t \times s$ is a *supermatrix* that consists of *submatrices* corresponding to $t' \times s'$:



Implementation 1.16 (`supermatrix`) *The* `supermatrix` *structure in the* C *programming language is implemented as follows:*

```
typedef struct _supermatrix supermatrix;
typedef supermatrix *psupermatrix;

struct _supermatrix {
  int rows;
  int cols;
  int block_rows;
  int block_cols;
  prkmatrix r;
  pfullmatrix f;
  psupermatrix* s;
};
```

A supermatrix M consists of `block_rows` $\times$ `block_cols` *submatrices. The size of the matrix is* `rows`$\times$`cols`*, i.e.,* $M \in \mathbb{R}^{\texttt{rows}\times\texttt{cols}}$*. The matrix can be*

- *an* `rkmatrix` *- then* `r` $\neq$ `0x0`*,* `f` $=$ `0x0` *and* `s` $=$ `0x0`*: the matrix* `r` *is the* `rkmatrix` *representation of M;*
- *a* `fullmatrix` *- then* `f` $\neq$ `0x0`*,* `r` $=$ `0x0` *and* `s` $=$ `0x0`*: the matrix* `f` *is the* `fullmatrix` *representation of M;*
- *a* `supermatrix` *- then* `s` $\neq$ `0x0`*,* `f` $=$ `0x0` *and* `r` $=$ `0x0`*: the array* `s` *contains the pointers to the submatrices $M_{i,j}$ of*

$$M = \begin{bmatrix} M_{1,1} & \cdots & M_{1,\texttt{blockcols}} \\ \vdots & \ddots & \vdots \\ M_{\texttt{blockrows},1} & \cdots & M_{\texttt{blockrows},\texttt{blockcols}} \end{bmatrix}$$

in the order

$$M_{1,1}, \ldots, M_{\texttt{blockrows},1}, \quad M_{1,2}, \ldots, M_{\texttt{blockrows},2}, \quad \ldots \quad , M_{1,\texttt{blockcols}}, \ldots, M_{\texttt{blockrows},\texttt{blockcols}},$$

so that the pointer to the submatrix $M_{i,j}$ is stored at the position `M->s[i+j*M->block_rows]`*.*

The implementation of an $\mathcal{H}$-matrix is a tree with nodes implemented as `supermatrix`. Additionally, the structure coincides with the structure given by the block cluster tree $T_{\mathcal{I}\times\mathcal{I}}$ (successors $\equiv$ submatrices) and the submatrices corresponding to admissible or inadmissible leaves are stored in the `rkmatrix` or `fullmatrix` format.

Chapter 2

Multi-dimensional Construction

In order to apply hierarchical matrices to problems in more than one space dimension, we have to introduce multi-dimensional counterparts of the cluster tree and the block cluster tree.

2.1 Multi-dimensional Cluster Tree

In the one-dimensional case, the clusters are organized in a balanced binary tree. For multi-dimensional problems, we need a generalized structure.

2.1.1 Definition and Basic Properties

Before we can define cluster trees, we have to define trees.

Definition 2.1 (Tree) *Let $N \neq \emptyset$ be a finite set, let $r \in N$ and let $S : N \to \mathcal{P}(N)$ be a mapping from N into subsets of N. For $t \in N$, a sequence $t_0, \ldots, t_m \in N$ with $t_0 = r$, $t_m = t$ and $t_{i+1} \in S(t_i)$ for all $i \in \{0, \ldots, m-1\}$ is called* sequence of ancestors *of t.*

$T := (N, r, S)$ is called a tree *if there is exactly one sequence of ancestors for each $t \in N$.*

If T is a tree, the elements of N are called nodes*, the element r is called the* root node *or* root *and denoted by* $\operatorname{root}(T)$*, and the set* $\operatorname{sons}(T, t) := S(t)$ *is called the* set of sons.

Lemma 2.2 (Properties of trees) *Let $T = (N, r, S)$ be a tree.*

1. *Let $t \in N$, and let $t_0, \ldots, t_m \in N$ be its sequence of ancestors. For all $i, j \in \{0, \ldots, m\}$ with $i \neq j$, we have $t_i \neq t_j$.*
2. *There is no $t \in N$ with $r \in \operatorname{sons}(t)$.*
3. *For each $t \in N \setminus \{r\}$, there is a unique $t^+ \in N$ with $t \in \operatorname{sons}(t^+)$.*

Proof: Let $i, j \in \{0, \ldots, m\}$ with $t_i = t_j$. Then both $t_0, \ldots, t_i$ and $t_0, \ldots, t_j$ are sequences of ancestors for $t_i = t_j$. Since these sequences are unique, we conclude $i = j$.

Let $t \in N$ with $\operatorname{sons}(T, t) \neq \emptyset$, and let $s \in \operatorname{sons}(T, t)$. Let $t_0, \ldots, t_m$ be the sequence of ancestors of t. Then $t_0, \ldots, t_m, s$ is a sequence of ancestors of s, and by definition it is the only one. This implies $r = t_0 \neq s$.

Let $t \in N \setminus \{r\}$, and let $t_0, \ldots, t_m$ be its sequence of ancestors. Due to $t \neq r = t_0$, we have $m > 0$ and find $t = t_m \in \operatorname{sons}(T, t_{m-1})$. Therefore we can conclude by setting $t^+ := t_{m-1}$. ∎

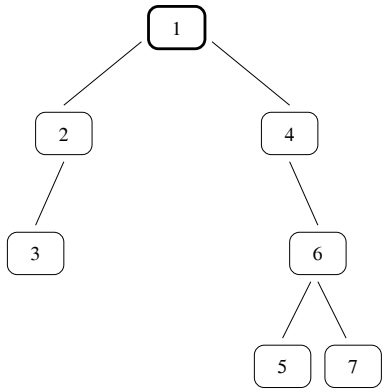


Figure 2.1: Tree in standard notation

Due to Lemma 2.2, for each $t \in N \setminus \{r\}$, there is a $t^+ \in N$ with $t \in \operatorname{sons}(t^+)$. This node is called the *father* of t and denoted by $\operatorname{father}(t) := t^+$.

We define the *set of descendants* $\operatorname{sons}^*(t)$ of a node $t \in N$ recursively by

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

A vertex $t \in N$ is a *leaf* if $\operatorname{sons}(t) = \emptyset$ holds and we define the *set of leaves*

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

Definition 2.3 (Tree level) *Let $T = (N, r, S)$ be a tree. Let $t \in N$, and let $t_0, \ldots, t_m \in N$ be its sequence of ancestors. The number $m \in \mathbb{N}_0$ is called the* level *of t and denoted by* $\operatorname{level}(T, t)$. *We define*

$$T^{(\ell)} := \{t \in N \ : \ \operatorname{level}(T, t) = \ell\} \qquad \text{for all } \ell \in \mathbb{N}_0.$$

We will use the short notations $t \in T$ instead of $t \in N$, $\operatorname{sons}(t)$ instead of $\operatorname{sons}(T, t)$ and $\operatorname{level}(t)$ instead of $\operatorname{level}(T, t)$ as long as this does not lead to confusion.

Obviously, we have $\operatorname{level}(t) = 0$ if and only if $t = \operatorname{root}(T)$. The maximal level is called the *depth* of T and denoted by

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

Definition 2.4 (Labeled tree) *Let $N, L \neq \emptyset$ be finite sets, let $r \in N$, let $S : N \to \mathcal{P}(N)$ and $m : N \to L$ be mappings. $T := (N, r, S, m, L)$ is a* labeled tree *if (N, r, S) is a tree.*

The notations for a tree carry over to a labeled tree. In addition, L is called the label set *of T and for each $t \in N$, $m(t) \in L$ is called the* label *of t and denoted by $\hat{t}$.*

Now we can generalize the structure introduced in the previous chapter: we organize subsets of the index set $\mathcal{I}$ in a cluster tree. The cluster tree supplies us with candidates that can be checked for admissibility. If they are not admissible, we split them and repeat the procedure. This suggests the following definition:

Definition 2.5 (Cluster tree) *A labeled tree $T = (N, r, S, m, L)$ is a* cluster tree *for an index set $\mathcal{I}$ if the following conditions hold:*

- $\widehat{\operatorname{root}(T)} = \mathcal{I}$.

- *Let $t \in N$. If* $\operatorname{sons}(T,t) \neq \emptyset$, *we have*
$$\hat{t} = \bigcup_{s \in \operatorname{sons}(t)} \hat{s}.$$
- *Let $t \in N$. For all $s_1, s_2 \in \operatorname{sons}(T,t)$ with $s_1 \neq s_2$, we have $\hat{s}_1 \cap \hat{s}_2 = \emptyset$.*

The vertices $t \in N$ of a cluster tree are called clusters.

A cluster tree for $\mathcal{I}$ is usually denoted by $T_{\mathcal{I}}$. We will use the abbreviation $t \in T_{\mathcal{I}}$ for $t \in N$.

Remark 2.6 (Alternative notation) *In the literature, the distinction between the cluster t and the corresponding index set $\hat{t}$ is frequently neglected. This simplifies the notation, but it can lead to ambiguities if clusters t with $\#\operatorname{sons}(T,t) = 1$ are present in the cluster tree: for $t' \in \operatorname{sons}(T,t)$, the definition of the cluster tree implies $\hat{t} = \hat{t}'$, although t and t' are different mathematical objects.*

Lemma 2.7 (Properties of cluster trees) *Let $T_{\mathcal{I}}$ be a cluster tree.*

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

Proof: We prove the first claim by induction over $\operatorname{level}(t) = \operatorname{level}(s) \in \mathbb{N}_0$. For $\operatorname{level}(t) = \operatorname{level}(s) = 0$, we have $t = \operatorname{root}(T_{\mathcal{I}}) = s$ and our statement is trivial. Let now $\ell \in \mathbb{N}_0$ be such that

$$t \neq s \Rightarrow \hat{t} \cap \hat{s} = \emptyset$$

holds for all $t, s \in T_{\mathcal{I}}$ with $\operatorname{level}(t) = \operatorname{level}(s) = \ell$.

Let $t, s \in T_{\mathcal{I}}$ with $t \neq s$ and $\operatorname{level}(t) = \operatorname{level}(s) = \ell + 1$. Since $\operatorname{level}(t) = \operatorname{level}(s) > 0$, there are clusters $t^+, s^+ \in T_{\mathcal{I}}$ with $t \in \operatorname{sons}(t^+)$, $s \in \operatorname{sons}(s^+)$ and $\operatorname{level}(t^+) = \operatorname{level}(t) - 1 = \ell = \operatorname{level}(s) - 1 = \operatorname{level}(s^+)$.

If $t^+ = s^+$, we have $s \in \operatorname{sons}(t^+)$, i.e., s and t are different sons of the same cluster, and Definition 2.5 implies $\hat{t} \cap \hat{s} = \emptyset$.

If $t^+ \neq s^+$, we can apply the induction assumption in order to find $\hat{t}^+ \cap \hat{s}^+ = \emptyset$. Definition 2.5 yields $\hat{t} \subseteq \hat{t}^+$ and $\hat{s} \subseteq \hat{s}^+$, which implies $\hat{t} \cap \hat{s} \subseteq \hat{t}^+ \cap \hat{s}^+ = \emptyset$ and concludes the induction.

Let us now consider the second claim. Let $t, s \in T_{\mathcal{I}}$ with $\operatorname{level}(t) \leq \operatorname{level}(s)$ and $\hat{t} \cap \hat{s} \neq \emptyset$. Let $m := \operatorname{level}(s)$ and $\ell := \operatorname{level}(t)$. Let $s_0, \ldots, s_m \in T_{\mathcal{I}}$ be the sequence of ancestors of s. Since $\ell \leq m$, $s^* := s_\ell$ is a well-defined cluster satisfying $s = s_m \in \operatorname{sons}^*(s^*)$ and $\operatorname{level}(s^*) = \ell = \operatorname{level}(t)$. Due to Definition 2.5, we have $\hat{s}^* \supseteq \hat{s}$ and therefore $\hat{s}^* \cap \hat{t} \supseteq \hat{s} \cap \hat{t} \neq \emptyset$. Using the first part of this proof, we can conclude $t = s^*$.

In order to prove the third claim, we let $i \in \mathcal{I}$ and introduce the set

$$C := \{t \in T_{\mathcal{I}} \ : \ i \in \hat{t}\}.$$

Due to $i \in \mathcal{I} = \widehat{\operatorname{root}(T_{\mathcal{I}})}$, we have $\operatorname{root}(T_{\mathcal{I}}) \in C$, i.e., the set C is not empty. Let $t \in C$. If $\operatorname{sons}(t) \neq \emptyset$, Definition 2.5 implies that there is a cluster $s \in \operatorname{sons}(t)$ with $i \in \hat{s} \subseteq \hat{t}$, which means $s \in C$. Let now $t \in C$ with $\operatorname{level}(t) = \max\{\operatorname{level}(s) \ : \ s \in C\}$. We have seen that the maximality of $\operatorname{level}(t)$ implies $\operatorname{sons}(t) = \emptyset$, therefore t is a leaf, and the definition of the set C implies $i \in \hat{t}$.

We can prove the fourth claim by combining the second and third claim: the third claim implies $\mathcal{I} = \bigcup_{t \in \mathcal{L}(T_{\mathcal{I}})} \hat{t}$. Let $t, s \in \mathcal{L}(T_{\mathcal{I}})$ with $t \neq s$. Without loss of generality, we can assume $\operatorname{level}(t) \leq \operatorname{level}(s)$. Since t is a leaf, we have $\operatorname{sons}^*(t) = \{t\}$, i.e., $s \notin \operatorname{sons}^*(t)$. Due to the second claim, this implies $\hat{t} \cap \hat{s} = \emptyset$. ∎

Exercise 1 *Let $T_{\mathcal{I}}$ be a cluster tree with c clusters and l leaves. Prove that $c \leq 2l - 1$ holds if we have $\#\operatorname{sons}(t) \neq 1$ for all $t \in T_{\mathcal{I}}$.*

Now that we know what a general cluster tree is, we have to find a suitable method for constructing a good cluster tree for a given set of basis functions. In the following, we will describe two simple algorithms that build cluster trees in a relatively general setting.

2.1.2 Geometric Bisection

The complexity of arithmetic operations for a hierarchical matrix is directly linked to the number of leaves of a block cluster tree, so a good cluster tree should make sure that blocks become admissible as soon as possible. The admissibility of a block depends on the diameters of the supports of the involved basis functions and on their distance. We cannot do much about the distance of the supports, but we can try to choose the cluster tree in such a way that the diameters shrink quickly.

For each $i \in \mathcal{I}$, we denote the support of the corresponding basis function φ_i by $\Omega_i := \operatorname{supp}(\varphi_i)$. Since dealing directly with the supports will be too complicated, we choose a point $x_i \in \Omega_i$ for each index $i \in \mathcal{I}$ and work with these points instead of the supports. This simplification will not significantly harm the performance of the algorithm, since the supports of typical finite element basis functions are small.

Our construction starts with the full index set $\mathcal{I}$, which is the root of the cluster tree by definition. Then, we apply a suitable technique to find a disjoint partition of the index set and use this partition to create son clusters. We apply the procedure recursively to the sons until the index sets are small enough.

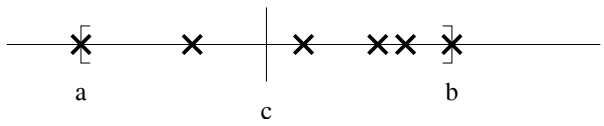
Let us first consider the one-dimensional case: We want to split the index set $\hat{t} \subset \mathcal{I}$ corresponding to a cluster t into two parts. Each index $i \in \hat{t}$ corresponds to a point $x_i \in \mathbb{R}$, so we can set $a := \min\{x_i \ : \ i \in \hat{t}\}$, $b := \max\{x_i \ : \ i \in \hat{t}\}$ and find $\{x_i \ : \ i \in \hat{t}\} \subseteq [a, b]$. Now the solution is clear: We set $c := (a + b)/2$ and split the interval $[a, b]$ into $[a, c]$ and $]c, b]$. This gives rise to the partition $\{\hat{t}_0, \hat{t}_1\}$ of $\hat{t}$ with

$$\hat{t}_0 := \{i \in \hat{t} \ : \ x_i \leq c\}, \quad \hat{t}_1 := \{i \in \hat{t} \ : \ x_i > c\}.$$

Obviously, we have

$$\left.\begin{matrix}\operatorname{diam}\{x_i \ : \ i \in \hat{t}_0\} \\ \operatorname{diam}\{x_i \ : \ i \in \hat{t}_1\}\end{matrix}\right\} \leq \frac{b-a}{2} = \frac{\operatorname{diam}\{x_i \ : \ i \in \hat{t}\}}{2},$$

so our choice is optimal.



In the multi-dimensional setting, we can generalize this approach: We set

$$a_l := \min\{(x_i)_l \ : \ i \in \hat{t}\} \quad \text{and} \quad b_l := \max\{(x_i)_l \ : \ i \in \hat{t}\}$$

for all $l \in \{1, \ldots, d\}$, therefore all points are contained in the axis-parallel box $[a_1, b_1] \times \cdots \times [a_d, b_d]$. Now we are faced with a choice: We can split the box in all coordinate directions simultaneously and get 2^d subdomains, or we can choose the coordinate direction of maximal extent and split the box perpendicular to this direction into two subdomains.

The first approach guarantees that the diameters of the subdomains are halved, but creates only a small number of clusters, so we will have only a small number of candidates to choose from in the construction of the block partition.

The second approach leads to a tree with a large number of clusters, but it also has the disadvantage that the diameters of the clusters will decrease only by a factor of $\sqrt{1 - 3/(4d)}$ during *one* step of the procedure, while performing d steps will still give us 2^d clusters with halved diameters, just as in the first approach.

2.1.3 Regular Subdivision

In some situations, we want to have a cluster structure that is more regular than the one resulting from standard geometric bisection, e.g., for the theoretical treatment of cluster algorithms or in situations where we want to use *one* cluster tree structure for different geometries.

As before, we construct the regular cluster tree by defining how a cluster t is split. We assume that a box $B_t = [a_1, b_1] \times \cdots \times [a_d, b_d]$ with $x_i \in B_t$ for all $i \in \hat{t}$ and a splitting direction $j_t \in \{1, \ldots, d\}$ are given. We construct new boxes B_{t_0} and B_{t_1} by setting $c_j := (a_j + b_j)/2$ and

$$B_{t_0} := [a_1, b_1] \times \cdots \times [a_j, c_j] \times \cdots \times [a_d, b_d] \quad \text{and} \quad B_{t_1} := [a_1, b_1] \times \cdots \times [c_j, b_j] \times \cdots \times [a_d, b_d].$$

The index sets $\hat{t}_0$ and $\hat{t}_1$ are defined by

$$\hat{t}_0 := \{i \in \hat{t} \ : \ x_i \in B_{t_0}\} \quad \text{and} \quad \hat{t}_1 := \hat{t} \setminus \hat{t}_1.$$

We set $j_{t_0} := j_{t_1} := (j_t \mod d) + 1$.

Since $x_i \in B_{t_0}$ for $i \in \hat{t}_0$ and $x_i \in B_{t_1}$ for $i \in \hat{t}_1$ hold by construction, we can now repeat the procedure for t_0 and t_1.

Compared to the standard geometric bisection, the regular construction has a major disadvantage: One of the son clusters $\hat{t}_1$ and $\hat{t}_2$ can be empty even if $\hat{t}$ is not, so it is possible to have clusters with exactly one son. But the regular construction also has a major advantage: all boxes on a level of the cluster tree have exactly the same dimensions, i.e., they are identical up to translations.

2.1.4 Implementation

The implementation of the tree structure is straightforward:

Implementation 2.8 (`cluster`) *The* `cluster` *structure is defined as follows:*

```
typedef struct _cluster cluster;
typedef cluster *pcluster;

struct _cluster {
  int start;
  int size;

  int sons;
  pcluster *son;
};
```

The fields `start` *and* `size` *describe* $\hat{t}$*: they give us the number of the first index of* $\hat{t}$ *and the number of indices. The meaning of these indices will become clear later.*

The field `sons` *contains the number of sons, i.e., the cardinality of the set* $\operatorname{sons}(t)$*, while the array* `son` *is filled with the pointers to these sons.*

We will now demonstrate how a cluster tree can be constructed. Before we start, we have to comment on a finer point of the implementation: in some applications, not all vertices appearing in a grid correspond to degrees of freedom. An example is the standard finite element method, applied to a problem with Dirichlet boundary conditions: the boundary vertices are present in the finite element grid, but they are not degrees of freedom.

In order to handle this in an elegant fashion, we distinguish *indices* and *degrees of freedom*. The indices form the contiguous subinterval $\{0, \ldots, n_{\mathrm{idx}} - 1\}$ of the set of integers, while the set $\mathcal{I}$ of degrees of freedom is an

arbitrary non-empty subset of the set of indices. In the finite element example, n_{idx} would be the number of *all* vertices, while $\mathcal{I}$ would be the subset containing non-boundary vertices.

Before we can give the complete algorithm, we first have to consider the necessary data structures: we need a `struct` to store the array of points and some auxiliary data:

Implementation 2.9 (`clusterfactory`) *The* `clusterfactory` *structure is defined as follows:*

```
typedef struct _clusterfactory clusterfactory;
typedef clusterfactory *pclusterfactory;

struct _clusterfactory {
  double **x;

  int ndof;
  int nidx;
  int d;

  double *vmin;
  double *vmax;

  double *blocks;
};
```

The field `ndof` *contains the maximal number of degrees of freedom, the field* `nidx` *contains the maximal number of points, while the field* `d` *gives us the spatial dimension of the underlying space.*

The array `x` *stores the coordinates of the points x_i for each index $i \in \{0, \ldots, \texttt{nidx-1}\}$: the entry* `x[i]` *is a* `d`*-dimensional array containing the coordinates of the point x_i.*

The fields `vmin` *and* `vmax` *will be used in our algorithms to store the minimal and maximal values corresponding to each coordinate.*

Since `x` *is implemented as an array of length* `nidx` *containing pointers to* `double` *variables, we need something these pointers can point to. Instead of allocating memory for each point individually, we allocated one block of size* `nidx*d` *and distribute this memory among the pointers* `x[i]`*. The field* `blocks` *points to the large block.*

The algorithm for the creation of a cluster tree is based on sets of indices. Therefore we have to find a suitable representation of sets and subsets in the C programming language.

We choose a simple approach: a set is an array of indices. The points are numbered by `int` quantities, so we represent a set by an array of `int` variables: the array

```
int index[4] = { 7, 2, 5, 8 };
```

corresponds to the set $\{7, 2, 5, 8\}$, the points can be accessed by `x[index[i]]` for values of `i` between `0` and `3`.

Obviously, the described set does not change if we rearrange the entries in the corresponding array:

```
int index2[4] = { 2, 5, 7, 8 };
```

describes exactly the same set as before. This fact can be used to treat subsets efficiently: if we want to split the set $\{7, 2, 5, 8\}$ into the subsets $\{2, 5\}$ and $\{7, 8\}$, we find that both subsets are described by two-element subarrays of `index2`, namely `index2` and `index2+2`.

This means that a clustering algorithm will give us two results: the cluster tree, expressed in the `cluster` structure, and an array `index` describing the mapping of degrees of freedom to indices. We collect these objects in a new class:

Implementation 2.10 (`clustertree`) *The* `clustertree` *structure is defined as follows:*

```
typedef struct _clustertree clustertree;
typedef clustertree *pclustertree;

struct _clustertree {
  int ndof;
  int nidx;

  int *dof2idx;
  int *idx2dof;
  pcluster root;
};
```

The field `ndof` *contains the maximal number of degrees of freedom, the field* `nidx` *contains the maximal number of points.*

The array `dof2idx` *has* `ndof` *elements and corresponds to the array* `index` *described above, i.e., it translates degrees of freedom into indices.*

The array `idx2dof` *has* `nidx` *elements and performs the inverse operation, i.e., it translates indices into degrees of freedom. The entries of indices that do not correspond to degrees of freedom are set to* `-1`*.*

The pointer `root` *gives us the root cluster of the cluster tree.*

Based on this representation of sets and cluster trees, the implementation of the geometric bisection clustering algorithm is straightforward: we use a recursive function `split_geometrically()` that receives a pointer `factory` to a `clusterfactory` structure containing the point coordinates, a pointer `index` to an `int` array describing the subset we have to split, and an `int size` giving us the size of this array. The function determines how to split the subset and rearranges the array `index` such that the first subset corresponds to the first part of the array and the second subset corresponds to the rest, then calls itself recursively.

In order to be able to reconstruct the subsets corresponding to all clusters, not only the leaf clusters, we add the additional `int` parameter `start` giving us the absolute starting index in the array describing the full index set. For performance reasons, we may want to stop the splitting process before we have reached sets containing only one element, so we also add an `int` parameter `leafsize` that tells us which sets are small enough.

```
static pcluster
split_geometrically(pclusterfactory factory, int *index,
                    int start, int sz, int leafsize)
{
  /* ... some initialization ... */

  if(sz <= leafsize)                    /* Stop if small enough */
    this = new_cluster(start, sz, 0);
  else {
    for(j=0; j<d; j++) {                /* Determine bounding box */
      vmin[j] = vmax[j] = x[index[0]][j];
      for(i=1; i<sz; i++)
        if(x[index[i]][j] < vmin[j])
          vmin[j] = x[index[i]][j];
```

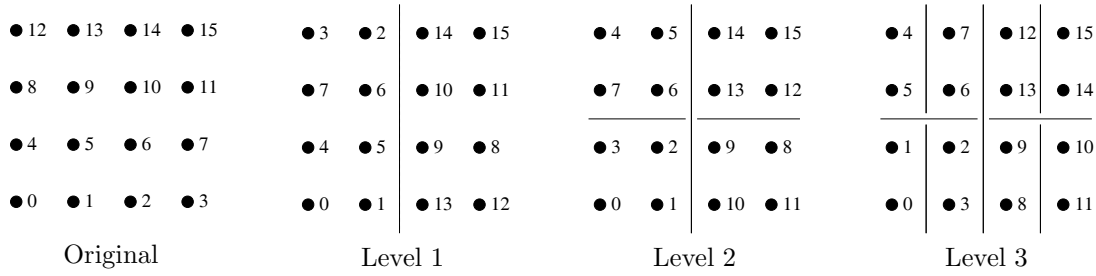
```
        else if(vmax[j] < x[index[i]][j])
          vmax[j] = x[index[i]][j];
      }

      jmax = 0; vdiff = vmax[0] - vmin[0]; /* Find maximal extent */
      for(j=1; j<d; j++)
        if(vmax[j] - vmin[j] > vdiff) {
          jmax = j; vdiff = vmax[j] - vmin[j];
        }

      l = 0; r = sz-1;                     /* Rearrange array */
      vmid = 0.5 * (vmax[jmax] + vmin[jmax]);
      while(l < r) {
        while(l < sz && x[index[l]][jmax] <= vmid) l++;
        while(r >= 0 && x[index[r]][jmax] >  vmid) r--;
        if(l < r) {
          h = index[l]; index[l] = index[r]; index[r] = h;
        }
      }

      this = new_cluster(start, sz, 2);    /* Recursion */
      this->son[0] = split_geometrically(factory, index,
                                         start, l, leafsize);
      this->son[1] = split_geometrically(factory, index+l,
                                         start+l, sz-l, leafsize);
    }

    return this;
  }
```

In order to simplify the handling of the creation of cluster trees, we employ a simple "front-end" procedure to call `split_geometrically`:

```
static pcluster
do_geometric(pclusterfactory factory,
             int *index, int n,
             int leafsize)
{
  return split_geometrically(factory, index, 0, n, leafsize);
}
```

For the clustering strategy based on regular subdivisions, we have to modify the splitting routine slightly: instead of computing the box given by `vmin` and `vmax` in each step, we simply update only one coordinate before calling the routine recursively.

If we apply one of these algorithms to a simple rectangular grid with lexicographic numbering of the degrees of freedom, we get the following sequence of numberings:



The array `index` maps the final numbering (i.e., the one corresponding to level 3) to the original numbering:

```
int index[16] = { 0, 4, 5, 1, 12, 8, 9, 13,
                  2, 6, 7, 3, 14, 10, 11, 15 };
```

In practice, the routines for the creation of cluster trees will be called not from the top-level of a program, but from intermediate routines that initialize the fields `x`, `smin` and `smax` with geometry data. Since we do not intend to write these routines for each type of clustering strategy, we use a "master function" that controls all aspects of the creation of a cluster tree:

```
pclustertree
create_subclustertree(pclusterfactory factory,
                      const int *index, int n,
                      ClusterStrategy strategy, int leafsize)
{
  /* ... some initialization ... */

  ct = new_clustertree(n, factory->nidx);
  dof2idx = ct->dof2idx;
  idx2dof = ct->idx2dof;

  for(i=0; i<n; i++) dof2idx[i] = index[i];

  switch(strategy) {
  default:
  case HLIB_DEFAULT:
  case HLIB_GEOMETRIC:
    ct->root = do_geometric(factory, dof2idx, n, leafsize);
    break;
  case HLIB_REGULAR:
    ct->root = do_regular(factory, dof2idx, n, leafsize);
    break;
  /* ... more clustering strategies ... */
  }

  for(i=0; i<factory->nidx; i++) idx2dof[i] = -1;
  for(i=0; i<n; i++) idx2dof[dof2idx[i]] = i;

  return ct;
}
```

Now let us turn our attention to a simple but non-trivial example that illustrates how we have to initialize the `clusterfactory` structure and how we can use it to create a cluster tree.

Example 2.11 (Curve in 2D) *We consider a closed curve in two-dimensional space, given as an array* `x` *of* `vertices` *points and an array* `e` *of* `edges` *edges. We use piecewise constant basis functions and choose the characterizing point to be the middle of the corresponding interval. Building the desired cluster tree is straightforward:*

```
factory = new_clusterfactory(edges, edges, 2);
index = (int *) malloc((size_t) sizeof(int) * edges);

for(i=0; i<edges; i++) {
  factory->x[i][0] = 0.5 * (x[e[i][0]][0] + x[e[i][1]][0]);
  factory->x[i][1] = 0.5 * (x[e[i][0]][1] + x[e[i][1]][1]);
  index[i] = i;
}

ct = create_subclustertree(factory, index, edges, HLIB_GEOMETRIC, 1);
```

In the previous example, each index corresponds to a degree of freedom. Now, we will consider a simple example in which the degrees of freedom are a true subset of the set of indices.

Example 2.12 (FE grid in 2D) *We consider a triangular grid, given as an array* `triangle` *of* `nt` *triangles, an array* `vertex` *of* `nv` *points and an array* `dirichlet` *of* `nv` *flags that are set if a point is part of the Dirichlet boundary. The construction of the cluster tree is done as follows:*

```
n = 0;
for(i=0; i<nv; i++)
  if(!dirichlet[i]) n++;

factory = new_clusterfactory(n, nv, 2);
index = (int *) malloc((size_t) sizeof(int) * n);

j = 0;
for(i=0; i<nv; i++) {
  factory->x[i][0] = vertex[i][0];
  factory->x[i][1] = vertex[i][1];
  if(!dirichlet[i]) {
    index[j] = i; j++;
  }
}

ct = create_subclustertree(factory, index, n, HLIB_REGULAR, 1);
```

For a given cluster `t`, we can iterate over all corresponding intervals by using the following simple loop:

```
for(i=t->start; i<t->start+t->size; i++) {
  ii = dof2idx[i];
  /* do something for the index ii */
}
```

For a pair `t`, `s` of clusters and a given rank `k`, we can build the `rkmatrix` by this function call:

```
rk = new_rkmatrix(k, t->size, s->size);
```

There is one very important detail: if we want to perform matrix operations, we use the permutation of the index set given in `dof2idx`, not the original ordering. The first index of a cluster `t` corresponds to the basis function with the number `dof2idx[t->start]`, not to that with the number `t->start`.

In typical applications, the permutation described by `dof2idx` is only important when accessing the underlying geometry, i.e., when matrices or right hand side vectors are discretized or when results have to be interpreted, e.g., if the solution vector is to be displayed.

Remark 2.13 (Alternatives) *There are many variations of the two possible clustering strategies described here: for some applications, a* balanced *cluster tree may be of importance, so the algorithm is changed so that the subsets it creates are of approximately identical size. For other applications, e.g., in boundary element techniques, it may be desirable to split the index set by more general planes than those given by the coordinate vectors we have used in our method.*

2.2 Multi-dimensional Block Cluster Tree

As seen in Subsection 1.5, the cluster trees can be used to derive a hierarchy of block partitions of the $\mathcal{I} \times \mathcal{J}$ corresponding to the matrix, the *block cluster tree*. The leaves of this tree form a *block partition* of $\mathcal{I} \times \mathcal{J}$.

2.2.1 Definition

Definition 2.14 (Block cluster tree) *Let $T_{\mathcal{I}}$ and $T_{\mathcal{J}}$ be cluster trees for index sets $\mathcal{I}$ and $\mathcal{J}$. A finite tree T is a* block cluster tree *for $T_{\mathcal{I}}$ and $T_{\mathcal{J}}$ if the following conditions hold:*

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

$$\operatorname{sons}(b) = \begin{cases} \{(t, s') \ : \ s' \in \operatorname{sons}(s)\} & \text{if } \operatorname{sons}(t) = \emptyset \text{ and } \operatorname{sons}(s) \neq \emptyset, \\ \{(t', s) \ : \ t' \in \operatorname{sons}(t)\} & \text{if } \operatorname{sons}(t) \neq \emptyset \text{ and } \operatorname{sons}(s) = \emptyset, \\ \{(t', s') \ : \ t' \in \operatorname{sons}(t), s' \in \operatorname{sons}(s)\} & \text{otherwise.} \end{cases} \tag{2.1}$$

- *The label of a node $b = (t, s) \in T$ is given by $\hat{b} = \hat{t} \times \hat{s} \subseteq \mathcal{I} \times \mathcal{J}$.*

The vertices of T are called block clusters.

A block cluster tree for $T_{\mathcal{I}}$ and $T_{\mathcal{J}}$ is usually denoted by $T_{\mathcal{I} \times \mathcal{J}}$.

We can see that $\widehat{\operatorname{root}(T_{\mathcal{I} \times \mathcal{J}})} = \mathcal{I} \times \mathcal{J}$ holds. A closer look at the definition reveals that $T_{\mathcal{I} \times \mathcal{J}}$ is a special cluster tree for the index set $\mathcal{I} \times \mathcal{J}$, therefore the leaves of $T_{\mathcal{I} \times \mathcal{J}}$ define a disjoint partition

$$\{\hat{b} \ : \ b \in \mathcal{L}(T_{\mathcal{I} \times \mathcal{J}})\} \tag{2.2}$$

of the index set $\mathcal{I} \times \mathcal{J}$ corresponding to a matrix.

Remark 2.15 (Alternative notation) *In the literature, block clusters are frequently denoted by $b = t \times s$ instead of $b = (t, s)$. The motivation for this notation is the one already pointed out in Remark 2.6: if t and $\hat{t}$ are treated as the same object, it is straightforward to also treat $b = (t, s)$ and $\hat{b} = \hat{t} \times \hat{s}$ as the same object. Using this sloppy notation, we have $b = \hat{b} = \hat{t} \times \hat{s} = t \times s$.*

Definition 2.16 (Level-consistency) *A block cluster tree $T_{\mathcal{I} \times \mathcal{J}}$ for $T_{\mathcal{I}}$ and $T_{\mathcal{J}}$ will be called* level-consistent*, if*

$$\operatorname{level}(b) = \operatorname{level}(t) = \operatorname{level}(s)$$

holds for all $b = (t, s) \in T_{\mathcal{I} \times \mathcal{J}}$.

If a block cluster tree is level-consistent, we have

$$\operatorname{sons}(b) \neq \emptyset \quad \Rightarrow \quad \operatorname{sons}(b) = \{t' \times s' \ : \ t' \in \operatorname{sons}(t), s' \in \operatorname{sons}(s)\}$$

for all $b = (t, s) \in T_{\mathcal{I}\times\mathcal{J}}$, i.e., only the last choice in (2.1) can hold.

2.2.2 Admissibility

In the one-dimensional setting, we have used the simple condition (1.4) to determine whether we could approximate a matrix block by a low-rank matrix.

In the multi-dimensional setting, we have to generalize the *admissibility condition*: the indices in $\mathcal{I}$ and therefore the clusters in $T_{\mathcal{I}}$ no longer correspond to intervals. We can still find a connection between indices and domains: each index $i \in \mathcal{I}$ corresponds to a basis function φ_i, and the support $\Omega_i = \operatorname{supp} \varphi_i$ again is a subdomain of $\mathbb{R}^d$.

We can generalize Ω_i to clusters $t \in T_{\mathcal{I}}$ by setting

$$\Omega_t := \bigcup_{i\in\hat{t}} \Omega_i,$$

i.e., Ω_t is the minimal subset of $\mathbb{R}^d$ that contains the supports of all basis functions φ_i with $i \in \hat{t}$.

Using this, a possible generalization of (1.4) is

$$\min\{\operatorname{diam}(\Omega_t), \operatorname{diam}(\Omega_s)\} \leq \eta \operatorname{dist}(\Omega_t, \Omega_s), \tag{2.3}$$

where $\operatorname{diam}(\cdot)$ is the Euclidean diameter of a set and $\operatorname{dist}(\cdot,\cdot)$ is the Euclidean distance of two sets.

The concept of admissibility carries over to clusters: a pair (t, s) of clusters $t \in T_{\mathcal{I}}$, $s \in T_{\mathcal{J}}$ is admissible if the corresponding domains Ω_t and Ω_s are admissible. Using this, we can define the necessary conditions for block cluster trees:

Definition 2.17 (Admissible block cluster tree) *A block cluster tree $T_{\mathcal{I}\times\mathcal{J}}$ for $\mathcal{I}$ and $\mathcal{J}$ is called* admissible *with respect to an admissibility condition if*

$$(t, s) \text{ is admissible} \quad \text{or} \quad \operatorname{sons}(t) = \emptyset \quad \text{or} \quad \operatorname{sons}(s) = \emptyset$$

holds for all leaves $(t, s) \in \mathcal{L}(T_{\mathcal{I}\times\mathcal{J}})$.

Constructing an admissible block cluster tree from the cluster trees $T_{\mathcal{I}}$ and $T_{\mathcal{J}}$ and a given admissibility condition can be done by a straightforward recursion: given two clusters $t \in T_{\mathcal{I}}$ and $s \in T_{\mathcal{J}}$, we check the admissibility. If the clusters are admissible, we are done. If they are not admissible, we repeat the procedure for all combinations of sons of t and sons of s.

Checking the condition (2.3) for general domains can be computationally expensive, so we are looking for a simplified condition. The traditional way is to determine the Chebyshev circles (in 2D) or spheres (in 3D) for the domains, since diameters and distances of circles or spheres can be computed in $\mathcal{O}(1)$ operations. Unfortunately, the construction of Chebyshev circles is not entirely trivial, so we make use of an even simpler technique: axis-parallel boxes.

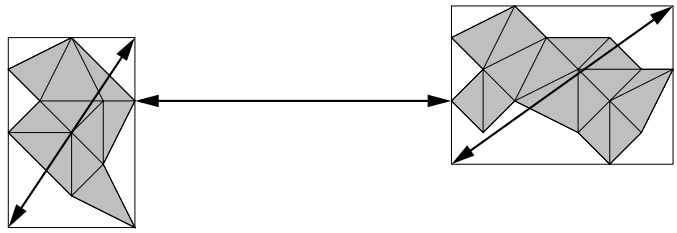
2.2.3 Bounding Boxes

For each cluster $t \in T_{\mathcal{I}}$, we define an axis-parallel box $Q_t \subseteq \mathbb{R}^d$ such that $\Omega_t \subseteq Q_t$ holds. This box will be called the *bounding box* of the cluster t.

By replacing the possibly complicated domains Ω_t and Ω_s in (2.3) by the larger boxes Q_t and Q_s, we get the admissibility condition

$$\min\{\operatorname{diam}(Q_t), \operatorname{diam}(Q_s)\} \leq \eta \operatorname{dist}(Q_t, Q_s). \tag{2.4}$$

This condition obviously implies (2.3).



Checking the distance and computing the diameters for axis-parallel boxes is not as simple as in the case of Chebyshev circles, but still manageable: if $Q_t = [a_1, b_1] \times \cdots \times [a_d, b_d]$ and $Q_s = [c_1, d_1] \times \cdots \times [c_d, d_d]$, we have

$$\operatorname{diam}(Q_t) = \left(\sum_{l=1}^{d} (b_l - a_l)^2 \right)^{1/2}, \quad \operatorname{diam}(Q_s) = \left(\sum_{l=1}^{d} (d_l - c_l)^2 \right)^{1/2} \quad \text{and}$$

$$\operatorname{dist}(Q_s, Q_s) = \left(\sum_{l=1}^{d} \operatorname{dist}([a_l, b_l], [c_l, d_l])^2 \right)^{1/2},$$

so these quantities can be computed by $\mathcal{O}(1)$ operations.

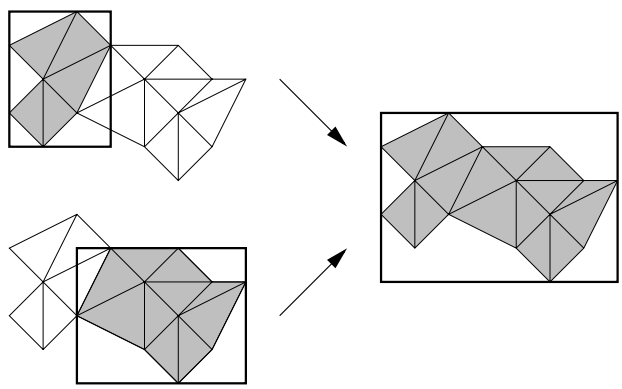
2.2.4 Implementation

Implementation 2.18 (Bounding boxes in `cluster`) *Since bounding boxes are fundamental, we modify the definition of the* `cluster` *structure by adding the necessary fields:*

```
double *bmin;
double *bmax;
int d;
```

The fields `bmin` *and* `bmax` *are* `d`*-dimensional arrays of* `double` *variables representing the minimal and maximal coordinates, i.e., if* $Q_t = [a_1, b_1] \times \cdots \times [a_d, b_d]$*, then* `bmin` *contains the vector* $(a_i)_{i=1}^d$ *and* `bmax` *contains the vector* $(b_i)_{i=1}^d$*.*

We can make use of the structure of the cluster tree in order to construct optimal bounding boxes, i.e., bounding boxes with minimal diameter: since $\hat{t} = \bigcup_{t' \in \operatorname{sons}(t)} \hat{t}'$ holds by definition, we have $\Omega_t = \bigcup_{t' \in \operatorname{sons}(t)} \Omega_{t'}$, so the optimal bounding box for the cluster t has to contain all the optimal bounding boxes for its sons t' and can therefore be constructed by finding maxima and minima of the corresponding coordinate vectors.



In the leaf clusters, we need information on the supports of the basis functions corresponding to the indices, and this geometric information has to be provided by the user. The obvious solution is to store it in the `clusterfactory` structure:

Implementation 2.19 (Bounding boxes of supports in `clusterfactory`) *We add the following two fields to the* `clusterfactory` *structure:*

```
double **smin;
double **smax;
```

The arrays `smin` *and* `smax` *have* `nidx` *entries, one for each index. The entries* `smin[i]` *and* `smax[i]` *describe the axis-parallel box containing the support Ω_i of the corresponding basis function in the same way the fields* `bmin` *and* `bmax` *describe the bounding box of a cluster.*

Example 2.20 (Curve in 2D, support) *In the Example 2.11, we have to add code that initializes the new fields* `smin` *and* `smax` *describing the bounding boxes of the supports of basis functions. Since these supports are simple intervals, it is sufficient to determine the minimal and maximal coordinates of the endpoints:*

```
for(i=0; i<edges; i++) {
  factory->smin[i][0] = dmin(x[e[i][0]][0], x[e[i][1]][0]);
  factory->smax[i][0] = dmax(x[e[i][0]][0], x[e[i][1]][0]);
  factory->smin[i][1] = dmin(x[e[i][0]][1], x[e[i][1]][1]);
  factory->smax[i][1] = dmax(x[e[i][0]][1], x[e[i][1]][1]);
}
```

Here, we use the functions `dmin` *and* `dmax` *to compute the minimum and maximum of two* `double` *arguments.*

Example 2.21 (FE grid in 2D, support) *Creating bounding boxes in the setting of Example 2.12 is slightly more complicated, since more than one triangle contribute to the support of a nodal basis function. Therefore we compute bounding boxes for all triangles and combine them to form boxes for the vertices:*

```
for(i=0; i<nv; i++) {
  factory->smin[i][0] = factory->smax[i][0] = vertex[i][0];
  factory->smin[i][1] = factory->smax[i][1] = vertex[i][1];
}

for(i=0; i<nt; i++) {
  tmin[0] = dmin(     vertex[triangle[i][0]][0],
                 dmin(vertex[triangle[i][1]][0],
                      vertex[triangle[i][2]][0]));
  tmax[0] = dmax(     vertex[triangle[i][0]][0],
                 dmax(vertex[triangle[i][1]][0],
                      vertex[triangle[i][2]][0]));
  tmin[1] = dmin(     vertex[triangle[i][0]][1],
                 dmin(vertex[triangle[i][1]][1],
                      vertex[triangle[i][2]][1]));
  tmax[1] = dmax(     vertex[triangle[i][0]][1],
                 dmax(vertex[triangle[i][1]][1],
                      vertex[triangle[i][2]][1]));

  for(j=0; j<3; j++) {
    k = triangle[i][j];
    factory->smin[k][0] = dmin(factory->smin[k][0], tmin[0]);
    factory->smax[k][0] = dmax(factory->smax[k][0], tmax[0]);
    factory->smin[k][1] = dmin(factory->smin[k][1], tmin[1]);
    factory->smax[k][1] = dmax(factory->smax[k][1], tmax[1]);
  }
}
```

Note that we have to initialize the bounding boxes with "safe" values before entering the main loop.

Using the extended `clusterfactory`, we can construct the bounding boxes and initialize the fields `bmin` and `bmax` in the `cluster` structure by the following recursive procedure:

```
static void
find_boundingbox(pcluster tau, pcclusterfactory factory)
{
  /* ... some initialization ... */

  if(sons > 0) {
    for(i=0; i<sons; i++)
      find_boundingbox(tau->son[i], factory);

    for(j=0; j<d; j++) {
      bmin[j] = son[0]->bmin; bmax[j] = son[0]->bmax;
    }

    for(i=1; i<sons; i++)
      for(j=0; j<d; j++) {
        bmin[j] = dmin(bmin[j], son[i]->bmin[j]);
        bmax[j] = dmax(bmax[j], son[i]->bmax[j]);
      }
  }
  else {
    for(j=0; j<d; j++) {
      bmin[j] = smin[index[0]]; bmax[j] = smax[index[0]];
    }

    for(i=1; i<size; i++)
      for(j=0; j<d; j++) {
        bmin[j] = dmin(bmin[j], smin[index[i]][j]);
        bmax[j] = dmax(bmax[j], smax[index[i]][j]);
    }
  }
}
```

Now we can use the bounding boxes in order to construct a block cluster tree.

Implementation 2.22 (`blockcluster`) *The* `blockcluster` *structure is defined as follows:*

```
typedef struct _blockcluster blockcluster;
typedef blockcluster *pblockcluster;

typedef struct {
  unsigned weakadm : 1;
  unsigned minadm : 1;
  unsigned maxadm : 1;
} BlockType;

struct _blockcluster {
  pccluster row;
  pccluster col;
  BlockType type;
```

```
  pblockcluster *son;
  int block_rows;
  int block_cols;
};
```

The fields `row` *and* `col` *give the row and column clusters that form this block. If the block has sons, the array* `son` *contains* `block_rows*block_cols` *pointers to these sons. The pointer to son* (i, j) *can be found at position* `i+j*block_rows`*. The bitfield* `type` *contains information about the admissibility criteria this block satisfies: it has subfields* `weakadm`*,* `minadm` *and* `maxadm` *corresponding to weak admissibility, standard admissibility (2.4) and strong admissibility (9.8).*

The following simple algorithm constructs a `blockcluster` structure from `cluster` structures and the standard admissibility condition (2.4):

```
pblockcluster
build_blockcluster(pccluster row, pccluster col, double eta)
{
  /* ... some initialization ... */

  dist = distance_cluster(row, col);
  diam_row = diameter_cluster(row);
  diam_col = diameter_cluster(col);

  type.maxadm = type.minadm = type.weakadm = 0;

  if(diam_row < eta*dist || diam_col < eta*dist) {
    type.minadm = type.weakadm = 1;
    bc = new_blockcluster(row, col, 0, 0, type);
  }
  else if(row->sons > 0 && col->sons > 0) {
    bc = new_blockcluster(row, col, row->sons, col->sons, type);
    for(j=0; j<col->sons; j++)
      for(i=0; i<row->sons; i++)
        bc->son[i+j*bc->block_rows] =
           build_blockcluster(row->son[i], col->son[j], eta);
  }
  else
    bc = new_blockcluster(row, col, 0, 0, type);

  return bc;
}
```

Exercise 2 *The routine* `build_blockcluster` *stops splitting matrix blocks as soon as one of the corresponding clusters is a leaf.*

Write a routine `build_blockcluster_inhom` *that splits the blocks until* both *clusters are leaves. If a pair* (t, s) *is not admissible, we distinguish four cases:*

- *If* $\mathrm{sons}(t) \neq \emptyset$ *and* $\mathrm{sons}(s) \neq \emptyset$*, examine all pairs* (t', s') *with* $t' \in \mathrm{sons}(t)$ *and* $s' \in \mathrm{sons}(s)$*.*
- *If* $\mathrm{sons}(t) = \emptyset$ *and* $\mathrm{sons}(s) \neq \emptyset$*, examine all pairs* (t, s') *with* $s' \in \mathrm{sons}(s)$*.*
- *If* $\mathrm{sons}(t) \neq \emptyset$ *and* $\mathrm{sons}(s) = \emptyset$*, examine all pairs* (t', s) *with* $t' \in \mathrm{sons}(t)$*.*
- *If* $\mathrm{sons}(t) = \emptyset$ *and* $\mathrm{sons}(s) = \emptyset$*, create a full matrix.*

Differently from `build_blockcluster`*, this new routine may create a block cluster tree that is not level-consistent.*

Implementation 2.23 (Construction of `blockcluster` structures) *In the library, we use a more general routine*

```
pblockcluster
build_blockcluster(pccluster row, pccluster col,
                   BlockAdmissiblityCriterion adm,
                   BlockHomogeneity hom, double eta, int leafsize);
```

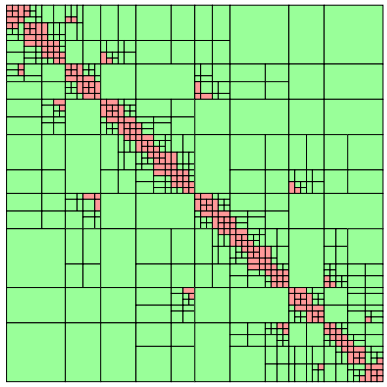
which allows the user to pick the desired kind of admissibility criterion `adm`*, to choose to create inhomogeneous block cluster trees (suited for $\mathcal{H}^2$-matrices) by setting* `hom` *appropriately, and to stop splitting clusters if they have not more than* `leafsize` *elements. The latter option is useful if a "coarse" block cluster tree has to be created without changing the* `clustertree` *structure.*

2.3 Construction of an Admissible `supermatrix` Structure

Since the structure of a `supermatrix` closely resembles that of a `blockcluster` tree, we can directly translate the latter into the former:

```
psupermatrix
build_supermatrix_from_blockcluster(pcblockcluster bc, int k)
{
  /* ... some initialization ... */

  if(bc->son) {
    s = new_supermatrix(block_rows, block_cols, rows, cols,
                        NULL, NULL, NULL);
    for(j=0; j<block_cols; j++)
      for(i=0; i<block_rows; i++)
        s->s[i+j*block_rows] =
          build_supermatrix_from_blockcluster(bc->son[i+j*block_rows],
                                              k, eps);
  }
  else {
    if(bc->type.weakadm) {
      r = new_rkmatrix(k, rows, cols);
      s = new_supermatrix(1, 1, rows, cols, NULL, r, NULL);
    }
    else {
      f = new_fullmatrix(rows, cols);
      s = new_supermatrix(1, 1, rows, cols, NULL, NULL, f);
    }
  }
  return s;
}
```

If we apply this algorithm to Example 2.11 and a block cluster tree constructed for the standard admissibility condition (2.4), we get a `supermatrix` structure that corresponds to the following matrix partition:



Chapter 3

Integral Equations

Using the multi-dimensional cluster tree and the corresponding generalised block cluster tree, we can store approximations of multi-dimensional discretise integral operators in the form of hierarchical matrices.

The basic approach is to replace the kernel by a degenerate expansion [44], this leads to the *panel-clustering method*. One of these expansions is the multipole expansion [53, 35] for the Laplace kernel, another one is the Taylor expansion which has already been used in Chapter 1.

3.1 Galerkin Discretisation

We consider a general Fredholm integral equation

$$\mathcal{G}[u] + \lambda u = f \tag{3.1}$$

in a Hilbert space H of functions on a domain Ω, where $\mathcal{G}$ is an integral operator

$$\mathcal{G}[u](x) = \int_\Omega g(x,y)u(y)\,\mathrm{d}x \tag{3.2}$$

corresponding to a kernel function g, which maps H into its dual space H'. The right hand side $f \in H'$ is an element of the dual space, $\lambda \in \mathbb{R}$ is some parameter and $u \in H$ is the solution we are looking for.

The variational counterpart of equation (3.1) is given by

$$a(u,v) + \lambda m(u,v) = \langle f, v\rangle_{H'\times H} \qquad \text{for all } v \in H, \tag{3.3}$$

where the bilinear forms a and m are given by

$$a(u,v) = \langle \mathcal{G}[u], v\rangle_{H'\times H} \quad \text{and} \quad m(u,v) = \langle u, v\rangle_{H\times H}.$$

The bilinear form $a(\cdot,\cdot)$ representing the integral operator can be written as

$$a(u,v) = \int_\Omega v(x) \int_\Omega g(x,y)u(y)\,\mathrm{d}y\,\mathrm{d}x. \tag{3.4}$$

The equation (3.3) is discretised by a Galerkin method, i.e., we choose an n-dimensional subspace H_n of H and consider the problem of finding a function $u_n \in H_n$ such that

$$a(u_n, v_n) + \lambda m(u_n, v_n) = \langle f, v_n\rangle_{H'\times H} \qquad \text{holds for all } v_n \in H_n.$$

Given a basis $(\varphi_i)_{i\in\mathcal{I}}$ of H_n, this is equivalent to

$$a(u_n, \varphi_i) + \lambda m(u_n, \varphi_i) = \langle f, \varphi_i\rangle_{H'\times H} \qquad \text{for all } i \in \mathcal{I}.$$

Since the solution u_n is an element of H_n, there is a coefficient vector $x = (x_i)_{i\in\mathcal{I}}$ satisfying

$$u_n = \sum_{j\in\mathcal{I}} x_j \varphi_j,$$

so the coefficients satisfy the equation

$$\sum_{j\in\mathcal{I}} x_j a(\varphi_j, \varphi_i) + \lambda \sum_{j\in\mathcal{I}} x_j m(\varphi_j, \varphi_i) = \langle f, \varphi_i \rangle_{H'\times H} \qquad \text{for all } i \in \mathcal{I}.$$

This is a system of linear equations and can be written in matrix form

$$Gx + \lambda M x = b$$

by introducing matrices $G, M \in \mathbb{R}^{\mathcal{I}\times\mathcal{I}}$ and a vector $b \in \mathbb{R}^{\mathcal{I}}$ with

$$G_{ij} := a(\varphi_j, \varphi_i) = \int_\Omega \varphi_i(x) \int_\Omega g(x,y)\varphi_j(y)\,\mathrm{d}y\,\mathrm{d}x \qquad \text{for all } i \in \mathcal{I}, j \in \mathcal{I}, \tag{3.5}$$

$$M_{ij} := m(\varphi_j, \varphi_i) = \int_\Omega \varphi_i(x)\varphi_j(x)\,\mathrm{d}x \qquad \text{for all } i \in \mathcal{I}, j \in \mathcal{I}, \text{ and} \tag{3.6}$$

$$b_i := \langle f, \varphi_i \rangle_{H'\times H} \qquad \text{for all } i \in \mathcal{I}. \tag{3.7}$$

If we use standard finite element basis functions $(\varphi_i)_{i\in\mathcal{I}}$, the matrix M will be sparse, but the matrix G will be densely populated, since typical kernel functions have global support.

Storing G directly will not lead to efficient algorithms. One way of avoiding this problem is to approximate G by a matrix that can be treated efficiently. The idea has already been described in Section 1.2: we replace the original kernel function $g(\cdot,\cdot)$ by local degenerate approximations, and this leads to a hierarchical matrix.

3.2 Degenerate Expansions

In Section 1.2, we have used the approximation

$$\tilde{g}(x,y) = \sum_{\nu=0}^{k-1} \frac{1}{\nu!} \partial_x^\nu g(x_0, y)(x - x_0)^\nu \tag{3.8}$$

of the kernel function $g(\cdot,\cdot)$. This approach can be generalised to the higher-dimensional case, but it will only work satisfactorily if the derivatives of the kernel function can be evaluated efficiently: in some cases it is possible to derive efficient recursion schemes to reach this goal, in other cases the kernel function is simply too complicated.

Fortunately, we are not forced to rely on Taylor expansions to construct $\mathcal{H}$-matrix approximations, we can use any approximation scheme which has two key properties: the approximant has to be *degenerate*, i.e., the variables x and y have to be separated, and it has to *converge rapidly* (cf. (1.5)) to the original kernel function g.

Before we consider concrete approximation schemes matching this description, let us investigate the general case: let $\tau, \sigma \subseteq \Omega$, let K be a general index set of cardinality k, and let $(v_\nu)_{\nu\in K}$ and $(w_\nu)_{\nu\in K}$ be functions on τ and σ, respectively, which satisfy

$$|g(x,y) - \tilde{g}(x,y)| \leq \varepsilon \qquad \text{for all } x \in \tau, y \in \sigma \tag{3.9}$$

for the general degenerate approximation

$$\tilde{g}(x,y) := \sum_{\nu\in K} v_\nu(x) w_\nu(y) \tag{3.10}$$

and a given error tolerance $\varepsilon \in \mathbb{R}_{>0}$.

Let t and s be clusters whose supports Ω_t and Ω_s satisfy

$$\Omega_t \subseteq \tau \quad \text{and} \quad \Omega_s \subseteq \sigma.$$

Then a rank-k-approximation of the subblock $G|_{\hat{t}\times\hat{s}}$ is given by

$$\begin{aligned}\tilde{G}^{t,s}_{ij} &:= \int_\Omega \varphi_i(x) \int_\Omega \tilde{g}(x,y)\varphi_j(y)\,\mathrm{d}x\,\mathrm{d}y = \sum_{\nu\in K} \int_\Omega \varphi_i(x) \int_\Omega v_\nu(x) w_\nu(y)\varphi_j(y)\,\mathrm{d}x\,\mathrm{d}y \\ &= \sum_{\nu\in K} \underbrace{\int_\Omega v_\nu(x)\varphi_i(x)\,\mathrm{d}x}_{=A_{i\nu}} \underbrace{\int_\Omega w_\nu(y)\varphi_j(y)\,\mathrm{d}y}_{=B_{j\nu}} = (AB^\top)_{ij} \qquad \text{for all } i \in \hat{t}, j \in \hat{s},\end{aligned}$$

where the matrices $A \in \mathbb{R}^{\hat{t}\times K}$ and $B \in \mathbb{R}^{\hat{s}\times K}$ are given by

$$A_{i\nu} := \int_\Omega v_\nu(x)\varphi_i(x)\,\mathrm{d}x \qquad \text{for all } i \in \hat{t}, \nu \in K, \tag{3.11}$$

$$B_{j\nu} := \int_\Omega w_\nu(y)\varphi_j(y)\,\mathrm{d}y \qquad \text{for all } j \in \hat{s}, \nu \in K. \tag{3.12}$$

Obviously, $\tilde{G}^{t,s} = AB^\top$ is given in `rkmatrix` representation, and this implies that its rank is bounded by $\#K = k$.

Now let us investigate the approximation error introduced by replacing $G|_{\hat{t}\times\hat{s}}$ by $\tilde{G}^{t,s}$. For each cluster $t \in T_{\mathcal{I}}$, we introduce the local coordinate isomorphism

$$P_t : \mathbb{R}^{\hat{t}} \to H, \qquad u \mapsto \sum_{i\in\hat{t}} u_i\varphi_i,$$

and a corresponding L^2-like norm

$$\|\cdot\|_{H_n} : \mathbb{R}^{\hat{t}} \to \mathbb{R}_{\geq 0}, \qquad u \mapsto \|P_t u\|_{L^2},$$

and find the following estimate:

Lemma 3.1 (Kernel approximation) *Let $t, s \in T_{\mathcal{I}}$ be clusters satisfying $\Omega_t \subseteq \tau$ and $\Omega_s \subseteq \sigma$, and let (3.9) hold. Then we have*

$$\langle v, (G|_{\hat{t}\times\hat{s}} - \tilde{G}^{t,s})u\rangle_2 \leq \varepsilon |\Omega_t|^{1/2}|\Omega_s|^{1/2}\|v\|_{H_n}\|u\|_{H_n} \qquad \text{for all } v \in \mathbb{R}^{\hat{t}}, u \in \mathbb{R}^{\hat{s}}.$$

Proof: Let $v \in \mathbb{R}^{\hat{t}}$ and $u \in \mathbb{R}^{\hat{s}}$. For $\hat{v} := P_t v$ and $\hat{u} := P_s u$, we find

$$\begin{aligned}|\langle v, (G|_{\hat{t}\times\hat{s}} - \tilde{G}^{t,s})u\rangle_2| &= \left|\int_\Omega \hat{v}(x) \int_\Omega (g(x,y) - \tilde{g}(x,y))\hat{u}(y)\,\mathrm{d}y\,\mathrm{d}x\right| \\ &\leq \int_\Omega |\hat{v}(x)| \int_\Omega |g(x,y) - \tilde{g}(x,y)||\hat{u}(y)|\,\mathrm{d}y\,\mathrm{d}x \\ &\leq \varepsilon \int_\Omega |\hat{v}(x)|\,\mathrm{d}x \int_\Omega |\hat{u}(y)|\,\mathrm{d}y = \varepsilon \int_{\Omega_t} |\hat{v}(x)|\,\mathrm{d}x \int_{\Omega_s} |\hat{u}(y)|\,\mathrm{d}y \\ &= \varepsilon \left(\int_{\Omega_t} \mathrm{d}x\right)^{1/2} \left(\int_{\Omega_t} \hat{v}^2(x)\,\mathrm{d}x\right)^{1/2} \left(\int_{\Omega_s} \mathrm{d}y\right)^{1/2} \left(\int_{\Omega_s} \hat{u}^2(y)\,\mathrm{d}y\right)^{1/2} \\ &\leq \varepsilon |\Omega_t|^{1/2}\|\hat{v}\|_{L^2}|\Omega_s|^{1/2}\|\hat{u}\|_{L^2} = \varepsilon |\Omega_t|^{1/2}|\Omega_s|^{1/2}\|v\|_{H_n}\|u\|_{H_n}.\end{aligned}$$

■

This error estimate demonstrates that a good approximation of g, i.e., a small value of ε, directly translates into a good approximation of $G|_{\hat{t}\times\hat{s}}$. Another important point is that the sizes $|\Omega_t|$ and $|\Omega_s|$ of the supports of the clusters t and s enter the estimate, since this suggests that only large clusters require a high accuracy, i.e., a higher rank, while small clusters can be handled with lower rank.

The estimate provided by Lemma 3.1 is not "purely algebraic", since it explicitly involves the coordinate isomorphisms, i.e., the discretisation scheme. Under the assumption that these isomorphisms are well-behaved, we can derive a more familiar-looking error estimate:

Lemma 3.2 (Block approximation) *Let $\mu_{\max} \in \mathbb{R}_{>0}$ be a constant satisfying*

$$\|u\|_{H_n}^2 \leq \mu_{\max}\|u\|_2^2 \qquad \text{for all } t \in T_{\mathcal{I}} \text{ and } u \in \mathbb{R}^{\hat{t}}.$$

Let $t, s \in T_{\mathcal{I}}$ be clusters satisfying $\Omega_t \subseteq \tau$ and $\Omega_s \subseteq \sigma$, and let (3.9) hold. Then we have

$$\|G|_{\hat{t}\times\hat{s}} - \tilde{G}^{t,s}\|_2 \leq \mu_{\max}\varepsilon|\Omega_t|^{1/2}|\Omega_s|^{1/2}. \tag{3.13}$$

Proof: Let $v \in \mathbb{R}^{\hat{t}}$ and $u \in \mathbb{R}^{\hat{s}}$. Due to Lemma 3.1, we have

$$|\langle v, (G|_{\hat{t}\times\hat{s}} - \tilde{G}^{t,s})u\rangle_2| \leq \varepsilon|\Omega_t|^{1/2}|\Omega_s|^{1/2}\|v\|_{H_n}\|u\|_{H_n} \leq \mu_{\max}\varepsilon|\Omega_t|^{1/2}|\Omega_s|^{1/2}\|v\|_2\|u\|_2.$$

Choosing $v := (G|_{\hat{t}\times\hat{s}} - \tilde{G}^{t,s})u$ yields

$$\|(G|_{\hat{t}\times\hat{s}} - \tilde{G}^{t,s})u\|_2^2 \leq \mu_{\max}\varepsilon|\Omega_t|^{1/2}|\Omega_s|^{1/2}\|(G|_{\hat{t}\times\hat{s}} - \tilde{G}^{t,s})u\|_2\|u\|_2,$$

and we can infer

$$\|(G|_{\hat{t}\times\hat{s}} - \tilde{G}^{t,s})u\|_2 \leq \mu_{\max}\varepsilon|\Omega_t|^{1/2}|\Omega_s|^{1/2}\|u\|_2$$

for all $u \in \mathbb{R}^{\hat{s}}$, which implies our claim. ∎

Remark 3.3 *The optimal choice for the constant $\mu_{\max}$ is the spectral norm of the mass matrix $M \in \mathbb{R}^{\mathcal{I}\times\mathcal{I}}$ given by (3.6): let $r := \operatorname{root}(T_{\mathcal{I}})$. Since $M = P_r^* P_r$, we have*

$$\max\left\{\frac{\|P_r u\|_{L^2}^2}{\|u\|_2^2} \;:\; u \in \mathbb{R}^{\mathcal{I}} \setminus \{0\}\right\} = \max\left\{\frac{\langle Mu, u\rangle_2}{\|u\|_2^2} \;:\; u \in \mathbb{R}^{\mathcal{I}} \setminus \{0\}\right\} = \|M\|_2$$

due to the Courant-Fischer Minimax Theorem [21, Theorem 8.1.2]. This means that $\mu_{\max} = \|M\|_2$ is the optimal choice for the root cluster r of $T_{\mathcal{I}}$. For an arbitrary cluster $t \in T_{\mathcal{I}}$, we apply this theorem to the principal submatrix $M|_{\hat{t}\times\hat{t}} = P_t^* P_t$ of M and find that $\mu_{\max} = \|M\|_2$ is already optimal.

For a discretisation using standard finite elements on a family of d_Ω-dimensional quasi-uniform meshes with mesh parameter $h \in \mathbb{R}_{>0}$, there is a constant $C_{\mathrm{fe}} \in \mathbb{R}_{>0}$ satisfying

$$\mu_{\max} \leq C_{\mathrm{fe}} h^{d_\Omega}.$$

We can combine estimates of the type (3.13) in order to derive global error estimates for hierarchical matrix approximations:

Lemma 3.4 (Global operator norm) *Let $T_{\mathcal{I}\times\mathcal{J}}$ be a block cluster tree. We have*

$$\|X\|_2 \leq \left(\sum_{b=(t,s)\in\mathcal{L}(T_{\mathcal{I}\times\mathcal{J}})} \|X|_{\hat{t}\times\hat{s}}\|_2^2\right) \qquad \text{for all } X \in \mathbb{R}^{\mathcal{I}\times\mathcal{J}}.$$

Proof: Let $v \in \mathbb{R}^{\mathcal{I}}$ and $u \in \mathbb{R}^{\mathcal{J}}$. We have

$$\begin{aligned}|\langle v, Xu\rangle_2| &= \left|\sum_{(t,s)\in\mathcal{L}(T_{\mathcal{I}\times\mathcal{J}})} \langle v|_{\hat{t}}, X|_{\hat{t}\times\hat{s}} u|_{\hat{s}}\rangle_2\right| \leq \sum_{(t,s)\in\mathcal{L}(T_{\mathcal{I}\times\mathcal{J}})} |\langle v|_{\hat{t}}, X|_{\hat{t}\times\hat{s}} u|_{\hat{s}}\rangle_2| \\ &\leq \sum_{(t,s)\in\mathcal{L}(T_{\mathcal{I}\times\mathcal{J}})} \|X|_{\hat{t}\times\hat{s}}\|_2 \|v|_{\hat{t}}\|_2 \|u|_{\hat{s}}\|_2 \\ &\leq \left(\sum_{(t,s)\in\mathcal{L}(T_{\mathcal{I}\times\mathcal{J}})} \|X|_{\hat{t}\times\hat{s}}\|_2^2\right)^{1/2} \left(\sum_{(t,s)\in\mathcal{L}^+(T_{\mathcal{I}\times\mathcal{J}})} \|v|_{\hat{t}}\|_2^2 \|u|_{\hat{s}}\|_2^2\right)^{1/2}.\end{aligned}$$

Due to Lemma 2.7, we have

$$\sum_{t\times s\in\mathcal{L}(T_{\mathcal{I}\times\mathcal{I}})} \|v|_{\hat{t}}\|_2^2 \|u|_{\hat{s}}\|_2^2 = \sum_{t\times s\in\mathcal{L}(T_{\mathcal{I}\times\mathcal{I}})} \sum_{i\in\hat{t}} \sum_{j\in\hat{s}} v_i^2 u_j^2 = \sum_{i\in\mathcal{I}} \sum_{j\in\mathcal{I}} v_i^2 u_j^2 = \|v\|_2^2 \|u\|_2^2$$

and can conclude

$$|\langle v, Xu\rangle_2| \leq \left(\sum_{(t,s)\in\mathcal{L}(T_{\mathcal{I}\times\mathcal{I}})} \|X|_{\hat{t}\times\hat{s}}\|_2^2\right)^{1/2} \|v\|_2 \|u\|_2.$$

Proceeding as in the proof of Lemma 3.2 yields the desired estimate. ∎

Theorem 3.5 (Operator-norm error) *Let $C_{\mathrm{ov}} \in \mathbb{N}$ be a constant satisfying*

$$\#\{i \in \mathcal{I} \ : \ \varphi_i(x) \neq 0\} \leq C_{\mathrm{ov}} \qquad \text{for all } x \in \Omega.$$

Let $\varepsilon \in \mathbb{R}_{>0}$. Let $T_{\mathcal{I}\times\mathcal{I}}$ be an admissible block cluster tree. For each admissible leaf $(t,s) \in \mathcal{L}^+(T_{\mathcal{I}\times\mathcal{I}})$, let $\tilde{G}^{t,s} \in \mathbb{R}^{\hat{t}\times\hat{s}}$ be an approximation of $G|_{\hat{t}\times\hat{s}}$ satisfying (3.13), and let the global approximation $\tilde{G} \in \mathbb{R}^{\mathcal{I}\times\mathcal{I}}$ be defined by

$$\tilde{G}|_{\hat{t}\times\hat{s}} := \begin{cases} \tilde{G}^{t,s} & \text{if } (t,s) \text{ is admissible,} \\ G|_{\hat{t}\times\hat{s}} & \text{otherwise,} \end{cases} \qquad \text{for all leaves } (t,s) \in \mathcal{L}(T_{\mathcal{I}\times\mathcal{I}}).$$

Then we have

$$\|G - \tilde{G}\|_2 \leq C_{\mathrm{ov}} \mu_{\max} |\Omega| \varepsilon.$$

Proof: Applying Lemma 3.4 to the matrix $X := G - \tilde{G}$ and the block cluster tree $T_{\mathcal{I}\times\mathcal{I}}$ yields

$$\|G - \tilde{G}\|_2 \leq \left(\sum_{(t,s)\in\mathcal{L}^+(T_{\mathcal{I}\times\mathcal{I}})} \|G|_{\hat{t}\times\hat{s}} - \tilde{G}^{t,s}\|_2^2\right)^{1/2}.$$

Due to (3.13), we find

$$\|G - \tilde{G}\|_2 \leq \varepsilon \mu_{\max} \left(\sum_{(t,s)\in\mathcal{L}(T_{\mathcal{I}\times\mathcal{I}})} |\Omega_t|\,|\Omega_s|\right)^{1/2}.$$

The analysis of this sum is based on Lemma 2.7: it implies that $\mathcal{L}(T_{\mathcal{I}\times\mathcal{I}})$ corresponds to a disjoint partition of $\mathcal{I}\times\mathcal{I}$, and we find

$$\sum_{t\times s\in\mathcal{L}(T_{\mathcal{I}\times\mathcal{I}})} |\Omega_t||\Omega_s| \leq \sum_{t\times s\in\mathcal{L}(T_{\mathcal{I}\times\mathcal{I}})} \sum_{i\in\hat{t}} \sum_{j\in\hat{s}} |\Omega_i||\Omega_j| = \sum_{i\in\mathcal{I}} \sum_{j\in\mathcal{I}} |\Omega_i||\Omega_j| = \left(\sum_{i\in\mathcal{I}} |\Omega_i|\right)^2. \tag{3.14}$$

We introduce

$$\chi_i : \Omega \to \{0,1\}, \qquad x \mapsto \begin{cases} 1 & \text{if } \varphi_i(x) \neq 0, \\ 0 & \text{otherwise,} \end{cases} \qquad \text{for all } i \in \mathcal{I}.$$

Due to our assumption, we have

$$\sum_{i \in \mathcal{I}} \chi_i(x) \leq C_{\mathrm{ov}}$$

in all points $x \in \Omega$ and conclude

$$\sum_{i \in \mathcal{I}} |\Omega_i| = \sum_{i \in \mathcal{I}} \int_\Omega \chi_i(x) \, \mathrm{d}x = \int_\Omega \sum_{i \in \mathcal{I}} \chi_i(x) \, \mathrm{d}x \leq C_{\mathrm{ov}} |\Omega|. \tag{3.15}$$

Combining (3.14) and (3.15) yields the estimate. ∎

Remark 3.6 *If the space H_n is constructed by a standard finite element approach using a local trial space on a reference element, the optimal choice for the constant C_{ov} is the dimension of this local space. For piecewise constant functions we have $C_{\mathrm{ov}} = 1$, while the well-known piecewise linear basis functions on triangles lead to the choice $C_{\mathrm{ov}} = 3$.*

The error estimate of Lemma 3.4 will, in general not be optimal: if we consider the identity matrix $I \in \mathbb{R}^{n \times n}$ as an n-by-n block matrix with blocks of size 1×1, the Lemma yields an estimate of $\sqrt{n}$ for the norm, while obviously 1 would be correct. If we assume that the block cluster tree is level-consistent and satisfies a property closely related to the concept ot *sparsity* introduced in Definition 7.3, we can derive the following improved error estimate:

Theorem 3.7 *Let $T_{\mathcal{I} \times \mathcal{J}}$ be a level-consistent block cluster tree of depth p, and let*

$$P_\ell := \{ b = (t,s) \in \mathcal{L}(T_{\mathcal{I} \times \mathcal{J}}) \ : \ \mathrm{level}(t) = \ell \} \qquad \text{for all } \ell \in \{0, \ldots, p\}.$$

Similar to Definition 7.3, we let

$$C_{\mathrm{sp}}^* := \max\{ \max\{ \#\{ s \in T_{\mathcal{J}} \ : \ (t,s) \in \mathcal{L}(T_{\mathcal{I} \times \mathcal{J}}) \} \ : \ t \in T_{\mathcal{I}} \}, \tag{3.16}$$

$$\max\{ \#\{ t \in T_{\mathcal{I}} \ : \ (t,s) \in \mathcal{L}(T_{\mathcal{I} \times \mathcal{J}}) \} \ : \ s \in T_{\mathcal{J}} \} \}. \tag{3.17}$$

The following inequality holds for the global and the blockwise spectral norms:

$$\|X\|_2 \leq C_{\mathrm{sp}}^* \sum_{\ell=0}^{p} \max\{ \|X|_{\hat{t} \times \hat{s}}\|_2 \ : \ b = (t,s) \in P_\ell \} \qquad \text{for all } X \in \mathbb{R}^{\mathcal{I} \times \mathcal{J}}. \tag{3.18}$$

Here we use the convention $\max \emptyset = 0$.

Proof: Let $P := \mathcal{L}(T_{\mathcal{I} \times \mathcal{J}})$, let

$$\varepsilon_\ell := \max\{ \|X|_{\hat{t} \times \hat{s}}\|_2 \ : \ b = (t,s) \in P_\ell \} \qquad \text{for all } \ell \in \{0, \ldots, p\},$$

and let Let $v \in \mathbb{R}^{\mathcal{I}}$ and $u \in \mathcal{J}$. According to Lemma 2.7 and the fact that $T_{\mathcal{I} \times \mathcal{J}}$ is a cluster tree for $\mathcal{I} \times \mathcal{J}$,

$$\{ \hat{t} \times \hat{s} \ : \ b = (t,s) \in P \}$$

is a disjoint partition of $\mathcal{I} \times \mathcal{J}$ and we have

$$|\langle v, Xu \rangle_2| = \left| \sum_{b=(t,s) \in P} \langle v|_{\hat{t}}, X|_{\hat{t} \times \hat{s}} u|_{\hat{s}} \rangle_2 \right| \leq \sum_{b=(t,s) \in P} \|v|_{\hat{t}}\|_2 \|X|_{\hat{t} \times \hat{s}}\|_2 \|u|_{\hat{s}}\|_2$$

$$
\begin{aligned}
&\leq \sum_{b=(t,s)\in P} \varepsilon_{\mathrm{level}(b)} \|v|_{\hat{t}}\|_2 \|u|_{\hat{s}}\|_2 = \sum_{b=(t,s)\in P} \left(\varepsilon_{\mathrm{level}(t)}^{1/2} \|v|_{\hat{t}}\|_2\right) \left(\varepsilon_{\mathrm{level}(s)}^{1/2} \|u|_{\hat{s}}\|_2\right) \\
&\leq \left(\sum_{b=(t,s)\in P} \varepsilon_{\mathrm{level}(t)} \|v|_{\hat{t}}\|_2^2\right)^{1/2} \left(\sum_{b=(t,s)\in P} \varepsilon_{\mathrm{level}(s)} \|u|_{\hat{s}}\|_2^2\right)^{1/2} \\
&= \left(\sum_{\ell=0}^{p} \varepsilon_\ell \sum_{t\in T_{\mathcal{I}}^{(\ell)}} \|v|_{\hat{t}}\|_2^2 \, \#\{s\in T_{\mathcal{J}} \ : \ (t,s)\in P_\ell\}\right)^{1/2} \\
&\quad \left(\sum_{\ell=0}^{p} \varepsilon_\ell \sum_{s\in T_{\mathcal{J}}^{(\ell)}} \|u|_{\hat{s}}\|_2^2 \, \#\{t\in T_{\mathcal{I}} \ : \ (t,s)\in P_\ell\}\right)^{1/2}.
\end{aligned}
$$

Since $P_\ell \subseteq P$, we have

$$
\max\{\#\{s\in T_{\mathcal{J}} \ : \ (t,s)\in P_\ell\} \ : \ t\in T_{\mathcal{I}}\} \leq C_{\mathrm{sp}}^*, \qquad \max\{\#\{t\in T_{\mathcal{I}} \ : \ (t,s)\in P_\ell\} \ : \ s\in T_{\mathcal{J}}\} \leq C_{\mathrm{sp}}^*,
$$

and due to Lemma 2.7, we find that $\{\hat{t} \ : \ t\in T_{\mathcal{I}}^{(\ell)}\} \subseteq \mathcal{I}$ and $\{\hat{s} \ : \ s\in T_{\mathcal{J}}^{(\ell)}\} \subseteq \mathcal{J}$ hold, which implies

$$
\sum_{t\in T_{\mathcal{I}}^{(\ell)}} \|v|_{\hat{t}}\|_2^2 \leq \|v\|_2^2 \qquad\qquad \sum_{s\in T_{\mathcal{J}}^{(\ell)}} \|u|_{\hat{s}}\|_2^2 \leq \|u\|_2^2
$$

Combining both estimates yields

$$
|\langle v, Xu\rangle_2| \leq \left(\sum_{\ell=0}^{p} \varepsilon_\ell C_{\mathrm{sp}}^* \|v\|_2^2\right)^{1/2} \left(\sum_{\ell=0}^{p} \varepsilon_\ell C_{\mathrm{sp}}^* \|u\|_2^2\right)^{1/2} = C_{\mathrm{sp}}^* \left(\sum_{\ell=0}^{p} \varepsilon_\ell\right) \|v\|_2 \|u\|_2.
$$

In the case $Xu = 0$, (3.18) holds trivially. Otherwise, we let $v := Xu$, find

$$
\|Xu\|_2^2 = \langle Xu, Xu\rangle_2 = \langle v, Xu\rangle_2 \leq C_{\mathrm{sp}}^* \left(\sum_{\ell=0}^{p} \varepsilon_\ell\right) \|Xu\|_2 \|u\|_2,
$$

and divide by $\|Xu\|_2$ in order to obtain the desired result. ∎

3.3 Interpolation

A relatively general approach to the construction of degenerate approximations is based on polynomial interpolation and has been described in [8, 47]. Its implementation is quite simple and it can be applied to all *asymptotically smooth* (cf. (3.38)) kernel functions.

Let $(\xi_\nu)_{\nu\in K}$ be a family of interpolation points in τ, and let $(\mathcal{L}_\nu)_{\nu\in K}$ be the corresponding Lagrange polynomials satisfying

$$
\mathcal{L}_\nu(\xi_\mu) = \delta_{\nu,\mu}
$$

for all $\nu, \mu \in K$. We interpolate the (function-valued) function $x \mapsto g(x,\cdot)$ and get the interpolant

$$
\tilde{g}(x,y) := \sum_{\nu\in K} g(\xi_\nu, y)\mathcal{L}_\nu(x). \tag{3.19}
$$

This approximation is obviously degenerate, and it can be constructed without the need for derivatives of the kernel function.

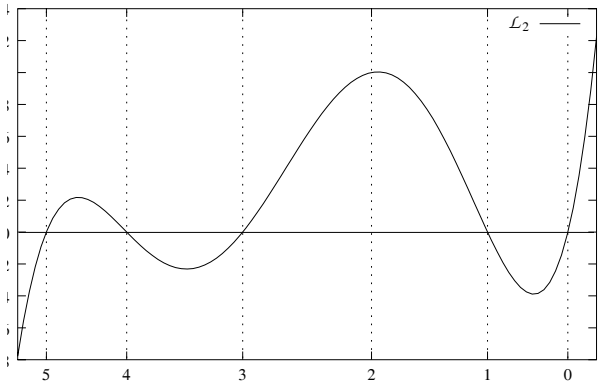


Figure 3.1: Lagrange polynomial $\mathcal{L}_2$ for $m = 5$

The computation of the entries of A and B defined in (3.11) and (3.12) is simple: the entries of the matrix A are given by

$$A_{i\nu} = \int_\Omega \mathcal{L}_\nu(x)\varphi_i(x)\,\mathrm{d}x \qquad \text{for all } i \in \hat{t}, \nu \in K, \tag{3.20}$$

i.e., if the basis functions φ_i are piecewise polynomial, the same will hold for the integrand, so that the entries can be computed by an exact quadrature rule. The entries of the matrix B are given by

$$B_{j\nu} = \int_\Omega g(\xi_\nu, y)\varphi_j(y)\,\mathrm{d}y \qquad \text{for all } j \in \hat{s}, \nu \in K, \tag{3.21}$$

and they can be computed either exactly (cf. Exercise 3) or approximated efficiently by quadrature rules: if the basis function is piecewise polynomial, the fact that the point x_ν is well separated from the support of φ_j implies that the integrand will be piecewise smooth.

3.3.1 Tensor-product Interpolation on Bounding Boxes

Since we cannot find a global degenerate approximation of the kernel function $g(\cdot,\cdot)$, we work with local approximations corresponding to pairs of clusters.

This means that we have to find a set of interpolation points and a set of corresponding Lagrange polynomials for each cluster $t \in T_\mathcal{I}$ such that the approximation error on the corresponding domain Ω_t is small enough.

Constructing good interpolation operators for general domains is a complicated topic, therefore we use the same simplification as in the treatment of the admissibility condition: instead of approximating the kernel function on a general subset Ω_t of $\mathbb{R}^d$, we approximate it on the bounding box $Q_t \supseteq \Omega_t$.

Since the bounding box Q_t is the tensor product of intervals, we first consider interpolation on intervals. For the interval $[-1, 1]$, the m-th order Chebyshev points

$$\xi_\nu := \cos\left(\frac{2\nu+1}{2m+2}\pi\right) \qquad \text{for all } \nu \in \{0,\ldots,m\}$$

are a good choice. The Lagrange polynomials (cf. Figure 3.1) have the form

$$\mathcal{L}_\nu(x) = \prod_{\mu=0,\mu\neq\nu}^{m} \frac{x-\xi_\mu}{\xi_\nu-\xi_\mu} \qquad \text{for all } \nu \in \{0,\ldots,m\}$$

and the corresponding interpolation operator is given by

$$\mathfrak{I}_m : C[-1,1] \to \mathcal{P}_m, \qquad f \mapsto \sum_{\nu=0}^{m} f(\xi_\nu)\mathcal{L}_\nu.$$

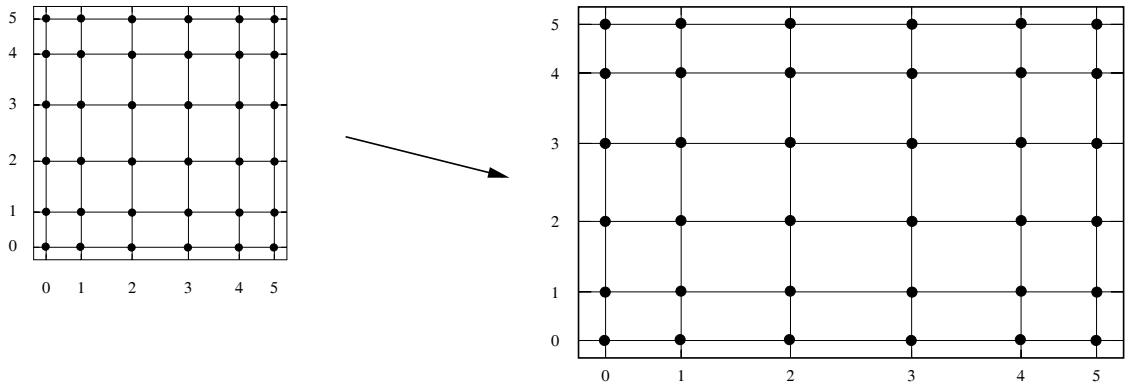


Figure 3.2: Transformation of two-dimensional tensor interpolation points

In order to get an interpolation operator for an arbitrary non-empty interval $[a, b]$, we use the bijective affine transformation

$$\Phi_{[a,b]} : [-1, 1] \to [a, b], \qquad x \mapsto \frac{b+a}{2} + \frac{b-a}{2}x$$

and define the transformed interpolation operator $\mathfrak{I}_m^{[a,b]} : C[a, b] \to \mathcal{P}_m$ by

$$\mathfrak{I}_m^{[a,b]}[f] := \big(\mathfrak{I}_m[f \circ \Phi_{[a,b]}]\big) \circ \Phi_{[a,b]}^{-1}. \tag{3.22}$$

It can be written in the form

$$\mathfrak{I}_m^{[a,b]}[f] = \sum_{\nu=0}^{m} f(\Phi_{[a,b]}(\xi_\nu))\mathcal{L}_\nu \circ \Phi_{[a,b]}^{-1},$$

so it is straightforward to define the transformed interpolation points

$$\xi_\nu^{[a,b]} := \Phi_{[a,b]}(\xi_\nu) = \frac{b+a}{2} + \frac{b-a}{2}\xi_\nu$$

and corresponding Lagrange functions

$$\mathcal{L}_\nu^{[a,b]} := \mathcal{L}_\nu \circ \Phi_{[a,b]}^{-1}.$$

We observe that

$$\mathcal{L}_\nu^{[a,b]}(\xi_\mu^{[a,b]}) = \mathcal{L}_\nu \circ \Phi_{[a,b]}^{-1}(\Phi_{[a,b]}(\xi_\mu)) = \mathcal{L}_\nu(\xi_\mu) = \delta_{\nu\mu}$$

holds for all $\nu, \mu \in \{0, \dots, m\}$. This implies

$$\mathcal{L}_\nu^{[a,b]}(x) = \prod_{\mu=0,\mu\neq\nu}^{m} \frac{x - \xi_\mu^{[a,b]}}{\xi_\nu^{[a,b]} - \xi_\mu^{[a,b]}} \qquad \text{for all } x \in [a, b].$$

In the d-dimensional case, the domain of interpolation is an axis-parallel bounding box $Q_t = [a_1, b_1] \times \cdots \times [a_d, b_d]$. Since the domain has tensor-product structure, it is straightforward to use tensor-product interpolation, i.e., to set

$$\mathfrak{I}_m^t := \mathfrak{I}_m^{[a_1,b_1]} \otimes \cdots \otimes \mathfrak{I}_m^{[a_d,b_d]}.$$

By introducing the set

$$K := \{\nu \in \mathbb{N}_0^d \ : \ \nu_i \leq m \text{ for all } i \in \{1, \dots, d\}\} = \{0, \dots, m\}^d$$

of multi-indices and the corresponding interpolation points (cf. Figure 3.2) and Lagrange polynomials

$$\xi_\nu^t := (\xi_{\nu_1}^{[a_1,b_1]}, \dots, \xi_{\nu_d}^{[a_d,b_d]}), \qquad \mathcal{L}_\nu^t := \mathcal{L}_{\nu_1}^{[a_1,b_1]} \otimes \cdots \otimes \mathcal{L}_{\nu_d}^{[a_d,b_d]}, \tag{3.23}$$

we can express $\mathfrak{I}_m^t$ in the familiar form

$$\mathfrak{I}_m^t[f](x) = \sum_{\nu \in K} f(\xi_\nu^t) \mathcal{L}_\nu^t(x).$$

Note that evaluating the tensor-product polynomials $\mathcal{L}_\nu^t$ is quite simple due to

$$\mathcal{L}_\nu^t(x) = \left(\mathcal{L}_{\nu_1}^{[a_1,b_1]} \otimes \cdots \otimes \mathcal{L}_{\nu_d}^{[a_d,b_d]}\right)(x) = \prod_{i=1}^d \mathcal{L}_{\nu_i}^{[a_i,b_i]}(x_i) = \prod_{i=1}^d \prod_{\mu=0,\mu\neq\nu_i}^m \frac{x_i - \xi_\mu^{[a_i,b_i]}}{\xi_{\nu_i}^{[a_i,b_i]} - \xi_\mu^{[a_i,b_i]}}. \tag{3.24}$$

3.3.2 Construction of the Low-rank Approximation

Let us consider an admissible pair (t, s) of clusters. The admissibility implies that

$$\min\{\operatorname{diam}(Q_t), \operatorname{diam}(Q_s)\} \leq \eta \operatorname{dist}(Q_t, Q_s)$$

holds. If $\operatorname{diam}(Q_t) \leq \operatorname{diam}(Q_s)$, we apply interpolation to the first argument of the kernel function. The corresponding block of the matrix has the form (3.19), so we have to compute the matrices $A^{t,s}$ and $B^{t,s}$:

$$A_{i\nu}^{t,s} = \int_\Omega \varphi_i(x) \mathcal{L}_\nu^t(x)\,\mathrm{d}x, \qquad B_{j\nu}^{t,s} = \int_\Omega \varphi_j(y) g(x_\nu^t, y)\,\mathrm{d}y,$$

where x_ν^t and $\mathcal{L}_\nu^t$ are the transformed interpolation points and Lagrange polynomials defined in (3.23).

If $\operatorname{diam}(Q_s) \leq \operatorname{diam}(Q_t)$, we apply interpolation to the second argument and have to compute the matrices $A^{t,s}$ and $B^{t,s}$ with reversed roles:

$$A_{i\nu}^{t,s} = \int_\Omega \varphi_i(x) g(x, x_\nu^s)\,\mathrm{d}x, \qquad B_{j\nu}^{t,s} = \int_\Omega \varphi_j(y) \mathcal{L}_\nu^s(y)\,\mathrm{d}y.$$

In both cases, we need the transformed interpolation points. Given an array `xp` of dimension `p` $= m+1$ containing the points $(\xi_i)_{i=0}^m$ and a cluster t, the following code fragment computes the corresponding transformed points and stores the points in an array `l` containing `p` entries containing the transformed points for each of the `d` dimensions:

```
for(j=0; j<d; j++) {
  mid = 0.5 * (t->bmax[j] + t->bmin[j]);
  dif = 0.5 * (t->bmax[j] - t->bmin[j]);

  for(i=0; i<p; i++)
    l[j][i] = mid + xp[i] * dif;
}
```

We store a multi-index $\nu \in \mathbb{N}_0^d$ in the form of an array `nu` of `d` integers. The j-th coordinate of the point ξ_ν^t is then given by `l[j][nu[j]]` for `nu[j]`$= \nu_j$. The computation of integrals of the Lagrange polynomials can be handled by exact quadrature rules, so only a way of evaluating a Lagrange polynomial corresponding to a multi-index ν at a point x is required. Due to (3.24), we can use the following simple code fragment:

```
result = 1.0;
for(j=0; j<d; j++)
  for(i=0; i<p; i++)
    if(i != nu[j])
      result *= (x[j] - l[j][i]) / (l[j][nu[j]] - l[j][i]);
```

3.3.3 Interpolation Error Bound

We will now investigate the error introduced by replacing the kernel function g by its interpolant $\tilde{g}$. We start by recalling the properties of one-dimensional interpolation, derive the properties of multi-dimensional tensor-product interpolation, and then apply these results to the problem of approximating g.

For each $m \in \mathbb{N}$, we let Λ_m be a constant satisfying the *stability estimate*

$$\|\mathfrak{I}_m[f]\|_{\infty,[-1,1]} \leq \Lambda_m \|f\|_{\infty,[-1,1]} \qquad \text{for all } f \in C[-1,1]. \tag{3.25}$$

In the case of Chebyshev interpolation, it is possible to show [51] that

$$\Lambda_m := \frac{2}{\pi}\ln(m+1) + 1 \leq m+1 \tag{3.26}$$

is a good choice for all $m \in \mathbb{N}$, i.e., the stability constant grows very slowly depending on m.

A bound for the approximation error can be derived by using the fact that $\mathfrak{I}_m$ is a projection onto the set $\mathcal{P}_m$ of m-th order polynomials, i.e., that we have

$$\mathfrak{I}_m[v] = v \qquad \text{for all } v \in \mathcal{P}_m. \tag{3.27}$$

Combining (3.27) with (3.25) yields $\Lambda_m \geq 1$.

Lemma 3.8 (Best-approximation property) *Let $\mathfrak{I}_m$ satisfy (3.25) and (3.27). Then we have*

$$\|f - \mathfrak{I}_m[f]\|_{\infty,[-1,1]} \leq (1+\Lambda_m)\inf\{\|f-v\|_{\infty,[-1,1]} \ : \ v \in \mathcal{P}_m\} \qquad \textit{for all } f \in C[-1,1]. \tag{3.28}$$

Proof: Let $f \in C[-1,1]$. For an arbitrary $v \in \mathcal{P}_m$, we can combine (3.27) and (3.25) in order to find

$$\begin{aligned}\|f - \mathfrak{I}_m[f]\|_{\infty,[-1,1]} &= \|f - v + \mathfrak{I}_m[v] - \mathfrak{I}_m[f]\|_{\infty,[-1,1]} = \|(f-v) - \mathfrak{I}_m[f-v]\|_{\infty,[-1,1]} \\ &\leq \|f-v\|_{\infty,[-1,1]} + \|\mathfrak{I}_m[f-v]\|_{\infty,[-1,1]} \\ &\leq (1+\Lambda_m)\|f-v\|_{\infty,[-1,1]}.\end{aligned}$$

Since v is arbitrary, this implies (3.28). ∎

This lemma demonstrates that a stable interpolation scheme yields the best approximation of any given function, up to the additional factor $1+\Lambda_m$, in the space of polynomials.

For a function $f \in C^{m+1}[-1,1]$, we can find a bound for the approximation error directly without relying on stability estimates: we get

$$\|f - \mathfrak{I}_m[f]\|_{\infty,[-1,1]} \leq \frac{2^{-m}}{(m+1)!}\|f^{(m+1)}\|_{\infty,[-1,1]}. \tag{3.29}$$

Let us now consider the transformed interpolation operator $\mathfrak{I}_m^{[a,b]}$ mapping $C[a,b]$ into $\mathcal{P}_m$. The stability estimate (3.25) yields

$$\begin{aligned}\|\mathfrak{I}_m^{[a,b]}[f]\|_{\infty,[a,b]} &= \|(\mathfrak{I}_m[f\circ\Phi_{[a,b]}])\circ\Phi_{[a,b]}^{-1}\|_{\infty,[a,b]} = \|\mathfrak{I}_m[f\circ\Phi_{[a,b]}]\|_{\infty,[-1,1]} \\ &\leq \Lambda_m\|f\circ\Phi_{[a,b]}\|_{\infty,[-1,1]} = \Lambda_m\|f\|_{\infty,[a,b]}\end{aligned} \tag{3.30}$$

for all $f \in C[a,b]$. The error estimate (3.29) takes the form

$$\begin{aligned}\|f - \mathfrak{I}_m^{[a,b]}[f]\|_{\infty,[a,b]} &= \|f\circ\Phi^{[a,b]} - (\mathfrak{I}_m^{[a,b]}[f])\circ\Phi^{[a,b]}\|_{\infty,[-1,1]} = \|f\circ\Phi^{[a,b]} - \mathfrak{I}_m[f\circ\Phi^{[a,b]}]\|_{\infty,[-1,1]} \\ &\leq \frac{2^{-m}}{(m+1)!}\|(f\circ\Phi^{[a,b]})^{(m+1)}\|_{\infty,[-1,1]} = \frac{2^{-m}}{(m+1)!}\left(\frac{b-a}{2}\right)^{m+1}\|f^{(m+1)}\|_{\infty,[a,b]} \\ &= \frac{2}{(m+1)!}\left(\frac{b-a}{4}\right)^{m+1}\|f^{(m+1)}\|_{\infty,[a,b]}\end{aligned} \tag{3.31}$$

for all $f \in C^{m+1}[a,b]$.

Now we will consider tensor-product interpolation operators on a general d-dimensional axis-parallel box $Q = [a_1,b_1] \times \cdots \times [a_d,b_d]$. Let

$$\mathfrak{I}_m^Q := \mathfrak{I}_m^{[a_1,b_1]} \otimes \cdots \otimes \mathfrak{I}_m^{[a_d,b_d]} \tag{3.32}$$

be the corresponding m-th order tensor-product interpolation operator mapping $C(Q)$ into $\mathcal{P}_m \otimes \cdots \otimes \mathcal{P}_m$. We introduce the componentwise interpolation operators

$$\mathfrak{I}_{m,i}^Q := I \otimes \cdots \otimes I \otimes \mathfrak{I}_m^{[a_i,b_i]} \otimes I \otimes \cdots \otimes I$$

for all $i \in \{1,\ldots,d\}$. For $i \in \{1,\ldots,d\}$ and a function $f \in C(Q)$, the function $\mathfrak{I}_{m,i}^Q[f]$ is a polynomial in the i-th coordinate:

$$\mathfrak{I}_{m,i}^Q[f](x) = \sum_{\nu=0}^{m} f(x_1,\ldots,x_{i-1},\xi_\nu^{[a_i,b_i]},x_{i+1},\ldots,x_d)\mathcal{L}_\nu^{[a_i,b_i]}(x_i) \qquad \text{for all } x \in Q. \tag{3.33}$$

Before we can investigate $\mathfrak{I}_m^Q$, we have to consider $\mathfrak{I}_{m,i}^Q$. Fortunately, the latter operator inherits most of its properties directly from the one-dimensional operator $\mathfrak{I}_m^{[a,b]}$:

Lemma 3.9 *Let $i \in \{1,\ldots,d\}$. We have*

$$\begin{aligned} \|\mathfrak{I}_{m,i}^Q[f]\|_{\infty,Q} &\leq \Lambda_m \|f\|_{\infty,Q} && \text{for all } f \in C(Q), \\ \|f - \mathfrak{I}_{m,i}^Q[f]\|_{\infty,Q} &\leq \frac{2}{(m+1)!}\left(\frac{b-a}{4}\right)^{m+1} \|\partial_i^{m+1} f\|_{\infty,Q} && \text{for all } f \in C^{m+1}(Q). \end{aligned}$$

Proof: Let $y_k \in [a_k,b_k]$ for $k \in \{1,\ldots,i-1,i+1,\ldots,m\}$. For all $f \in C(Q)$, we define the function

$$f_i : [a_i,b_i] \to \mathbb{R}, \qquad x \mapsto f(y_1,\ldots,y_{i-1},x,y_{i+1},\ldots,y_d)$$

and observe

$$\begin{aligned} \|f_i\|_{\infty,[a_i,b_i]} &\leq \|f\|_{\infty,Q} && \text{for all } f \in C(Q), \\ \|f_i^{(m+1)}\|_{\infty,[a_i,b_i]} &\leq \|\partial_i^{m+1} f\|_{\infty,Q} && \text{for all } f \in C^{m+1}(Q). \end{aligned}$$

Let now $f \in C(Q)$. We have

$$\begin{aligned} f(y_1,\ldots,y_{i-1},x,y_{i+1},\ldots,y_d) &= f_i(x), \\ \mathfrak{I}_{m,i}^Q[f](y_1,\ldots,y_{i-1},x,y_{i+1},\ldots,y_d) &= \mathfrak{I}_m^{[a_i,b_i]}[f_i](x), \end{aligned}$$

so the stability estimate (3.30) yields

$$\left|\mathfrak{I}_{m,i}^Q[f](y_1,\ldots,y_{i-1},x,y_{i+1},\ldots,y_d)\right| \leq \left|\mathfrak{I}_m^{[a_i,b_i]}[f_i](x)\right| \leq \Lambda_m \|f_i\|_{\infty,[a_i,b_i]} \leq \Lambda_m \|f\|_{\infty,Q}. \tag{3.34}$$

For $f \in C^{m+1}(Q)$, we can use the approximation error estimate (3.31) in order to find

$$\begin{aligned} &\left|f(y_1,\ldots,y_{i-1},x,y_{i+1},\ldots,y_d) - \mathfrak{I}_{m,i}^Q[f](y_1,\ldots,y_{i-1},x,y_{i+1},\ldots,y_d)\right| = \left|f_i(x) - \mathfrak{I}_m^{[a_i,b_i]}[f_i](x)\right| \\ &\quad \leq \frac{2}{(m+1)}\left(\frac{b_i-a_i}{4}\right)^{m+1} \|f_i^{(m+1)}\|_{\infty,[a_i,b_i]} \leq \frac{2}{(m+1)}\left(\frac{b_i-a_i}{4}\right)^{m+1} \|\partial_i^{m+1} f\|_{\infty,Q}. \end{aligned} \tag{3.35}$$

Applying (3.34) and (3.35) to arbitrary points $(y_1,\ldots,y_{i-1},x,y_{i+1},\ldots,y_d) \in Q$ yields the desired result. ■

Now that we have established the stability and approximation properties of $\mathfrak{I}_{m,i}^Q$, we can do the same for the tensor-product operator $\mathfrak{I}_m^Q$:

Theorem 3.10 (Tensor-product interpolation) *Let $Q = [a_1, b_1] \times \cdots \times [a_d, b_d]$, let $m \in \mathbb{N}$. Let $\mathfrak{I}_m$ satisfy the stability estimate (3.25) and the error estimate (3.29). Let $\mathfrak{I}_m^Q$ be the m-th order tensor-product interpolation operator defined by (3.32). We have*

$$\|\mathfrak{I}_m^Q[f]\|_{\infty,Q} \leq \Lambda_m^d \|f\|_{\infty,Q} \qquad \text{for all } f \in C(Q),$$
$$\|f - \mathfrak{I}_m^Q[f]\|_{\infty,Q} \leq 2d\Lambda_m^{d-1} \left(\frac{\operatorname{diam}(Q)}{4}\right)^{m+1} \frac{\max\{\|\partial_i^{m+1} f\|_{\infty,Q} \ : \ i \in \{1,\ldots,d\}\}}{(m+1)!} \qquad \text{for all } f \in C^{m+1}(Q).$$

Proof: For all $i \in \{0,\ldots,d\}$, we introduce the auxiliary operator

$$P_i := \mathfrak{I}_m^{[a_1,b_1]} \otimes \cdots \otimes \mathfrak{I}_m^{[a_i,b_i]} \otimes I \otimes \cdots \otimes I = \prod_{k=1}^{i} \mathfrak{I}_{m,k}^Q.$$

We observe $P_0 = I$ and $P_d = \mathfrak{I}_m^Q$. Let $i \in \{1,\ldots,d\}$. We have $P_i = \mathfrak{I}_{m,i}^Q P_{i-1}$, so the first inequality of Lemma 3.9 yields

$$\|P_i[f]\|_{\infty,Q} = \|\mathfrak{I}_{m,i}^Q[P_{i-1}[f]]\|_{\infty,Q} \leq \Lambda_m \|P_{i-1}[f]\|_{\infty,Q},$$

and we get

$$\|P_i[f]\|_{\infty,Q} \leq \Lambda_m^i \|f\|_{\infty,Q} \tag{3.36}$$

by induction. This implies

$$\|\mathfrak{I}_m^Q[f]\|_{\infty,Q} = \|P_d[f]\|_{\infty,Q} \leq \Lambda_m^d \|f\|_{\infty,Q},$$

which is the first inequality of Theorem 3.10. In order to prove the second one, we combine (3.36) with Lemma 3.9 and find

$$\begin{aligned}
\|f - \mathfrak{I}_m^Q f\|_{\infty,Q} &\leq \sum_{i=1}^{d} \|P_{i-1}f - P_i f\|_{\infty,Q} = \sum_{i=1}^{d} \|P_{i-1}[f - \mathfrak{I}_{m,i}^Q[f]]\|_{\infty,Q} \\
&\leq \sum_{i=1}^{d} \Lambda_m^{i-1} \|f - \mathfrak{I}_{m,i}^Q[f]\|_{\infty,Q} \leq \frac{2\Lambda_m^{d-1}}{(m+1)!} \sum_{i=1}^{d} \left(\frac{b_i - a_i}{4}\right)^{m+1} \|\partial_i^{m+1} f\|_{\infty,Q} \qquad (3.37) \\
&\leq \frac{2d\Lambda_m^{d-1}}{(m+1)!} \left(\frac{\operatorname{diam}(Q)}{4}\right)^{m+1} \max\{\|\partial_i^{m+1} f\|_{\infty,Q} \ : \ i \in \{1,\ldots,d\}\},
\end{aligned}$$

and this was to be demonstrated. ∎

Remark 3.11 (Anisotropic interpolation) *The intermediate estimate (3.37) in the proof of Theorem 3.10 suggests a modification of the interpolation scheme: if the dimensions of the bounding box vary, i.e., if it is short in some directions and long in other ones, it may be possible to use a lower interpolation order in the shorter directions without harming the total interpolation error.*

Now let us apply Theorem 3.10 to the kernel function g.

We assume that $g \in C^\infty(Q_t \times Q_s)$ is *asymptotically smooth*, i.e, that

$$|\partial_x^\alpha \partial_y^\beta g(x,y)| \leq C\,|\alpha+\beta|!\; c_0^{|\alpha+\beta|}\, \|x-y\|^{-|\alpha+\beta|-\sigma} \tag{3.38}$$

holds for constants $C, c_0, \sigma \in \mathbb{R}_{>0}$. This implies that the functions

$$\begin{aligned}
g_x &: Q_t \to C^\infty(Q_s), \quad x \mapsto g(x,\cdot), \\
g_y &: Q_s \to C^\infty(Q_t), \quad y \mapsto g(\cdot,y),
\end{aligned} \tag{3.39}$$

satisfy the estimates

$$\|\partial^\alpha g_x(x)\|_{\infty,Q_t} \leq C\,|\alpha|!\; c_0^{|\alpha|}\, \operatorname{dist}(Q_t,Q_s)^{-|\alpha|-\sigma},$$

$$\|\partial^\beta g_y(y)\|_{\infty,Q_s} \leq C\,|\beta|!\;c_0^{|\beta|}\;\operatorname{dist}(Q_t,Q_s)^{-|\beta|-\sigma}$$

for $x \in Q_t$ and $y \in Q_s$, so we can apply Theorem 3.10 and use $\Lambda_m \leq m+1$ in order to get

$$\|g_x - \mathfrak{I}_m^t[g_x]\|_{\infty,Q_t} \leq \frac{2Cd(m+1)^{d-1}}{\operatorname{dist}(Q_t,Q_s)^\sigma}\left(\frac{c_0\operatorname{diam}(Q_t)}{4\operatorname{dist}(Q_t,Q_s)}\right)^{m+1}$$
$$\|g_y - \mathfrak{I}_m^s[g_y]\|_{\infty,Q_s} \leq \frac{2Cd(m+1)^{d-1}}{\operatorname{dist}(Q_t,Q_s)^\sigma}\left(\frac{c_0\operatorname{diam}(Q_s)}{4\operatorname{dist}(Q_t,Q_s)}\right)^{m+1}.$$

The approximation $\tilde{g}$ is given by

$$\tilde{g}(x,y) := \begin{cases} (\mathfrak{I}_m^t[g_x])(x)(y) & \text{if } \operatorname{diam}(Q_t) \leq \operatorname{diam}(Q_s), \\ (\mathfrak{I}_m^s[g_y])(y)(x) & \text{otherwise} \end{cases} \qquad \text{for all } x \in Q_t, y \in Q_s. \tag{3.40}$$

If $\operatorname{diam}(Q_t) \leq \operatorname{diam}(Q_s)$, we have

$$|g(x,y) - \tilde{g}(x,y)| = |g(x,y) - \sum_{\nu\in K} g(x_\nu^t,y)\mathcal{L}_\nu^t(x)| = |g_x(x)(y) - (\mathfrak{I}_m^t[g_x](x))(y)| = |(g_x - \mathfrak{I}_m^t[g_x])(x)(y)|$$
$$\leq \frac{2Cd(m+1)^{d-1}}{\operatorname{dist}(Q_t,Q_s)^\sigma}\left(\frac{c_0\operatorname{diam}(Q_t)}{4\operatorname{dist}(Q_t,Q_s)}\right)^{m+1} \leq \frac{2Cd(m+1)^{d-1}\eta^\sigma}{\min\{\operatorname{diam}(Q_t),\operatorname{diam}(Q_s)\}^\sigma}\left(\frac{c_0\eta}{4}\right)^{m+1}.$$

In a similar fashion, we can treat the case $\operatorname{diam}(Q_s) \leq \operatorname{diam}(Q_t)$ and conclude

$$|g(x,y) - \tilde{g}(x,y)| \leq \frac{C_{\text{in}}(m)}{\operatorname{dist}(Q_t,Q_s)^\sigma}\left(\frac{c_0\eta}{4}\right)^{m+1} \tag{3.41}$$

for the polynomial $C_{\text{in}}(m) := 2Cd(m+1)^d\eta^\sigma$. If we choose $\eta < 4/c_0$, we have $(c_0\eta/4) < 1$ and the approximation of the kernel function converges exponentially in m.

Remark 3.12 (Improved error bound) *In [8], we prove that the error converges as $\mathcal{O}((c_0\eta)/(c_0\eta+2))$, i.e., we get exponential convergence even if η and c_0 are large.*

3.4 Approximation of Derivatives

In the context of boundary element methods, we are frequently faced by the challenge of approximating products of derivatives of an asymptotically smooth function and another function which is not asymptotically smooth, but separable.

A typical example is the classical double layer potential operator

$$\mathcal{G}_{\text{DLP}}[u](x) = \frac{1}{2\pi}\int_\Gamma \frac{\langle x-y, n(y)\rangle}{\|x-y\|^2}$$

on a curve Γ in two spatial dimensions, where for each $y \in \Gamma$ the unit vector $n(y)$ describes the outward normal direction. The kernel function

$$g(x,y) = \frac{1}{2\pi}\frac{\langle x-y, n(y)\rangle}{\|x-y\|^2}$$

is not asymptotically smooth: n is only defined on Γ, and it will, in general, not be smooth. Therefore the theory of the preceding section does not apply directly to this integral operator.

3.4.1 Separable Approximation of Partial Derivatives

We observe that we have

$$g(x,y) = \langle \operatorname{grad}_y \gamma(x,y), n(y) \rangle \qquad \text{for } x, y \in \mathbb{R}^2, x \neq y$$

for the *generator function*

$$\gamma : (\mathbb{R}^2) \times (\mathbb{R}^2) \to \mathbb{R}, \qquad (x,y) \mapsto \begin{cases} -\frac{1}{2\pi} \log \|x-y\| & \text{if } x \neq y, \\ 0 & \text{otherwise.} \end{cases}$$

This function is asymptotically smooth, so g is the product of a derivative of γ and a separable function (since n does not depend on x, it is trivially separable).

We can apply the theory of the previous sections to γ in order to construct a separable approximation

$$\tilde{\gamma}(x,y) = \sum_{\nu \in K} v_\nu(x) w_\nu(y),$$

which gives rise to a separable approximation

$$\tilde{g}(x,y) := \langle \operatorname{grad}_y \tilde{\gamma}(x,y), n(y) \rangle = \sum_{\nu \in K} v_\nu(x) \langle \operatorname{grad} w_\nu(y), n(y) \rangle$$

of the kernel function g. Using this approach, we can construct low-rank approximations of matrix blocks by the techniques we have already introduced.

Now let us investigate the error introduced by replacing g by $\tilde{g}$. We have

$$|g(x,y) - \tilde{g}(x,y)| = |\langle \operatorname{grad}_y(\gamma - \tilde{\gamma})(x,y), n(y) \rangle \leq \| \operatorname{grad}_y(\gamma - \tilde{\gamma})(x,y) \| \, \|n(y)\|$$

and due to $\|n(y)\| = 1$, we are left with the task of finding a bound for $\| \operatorname{grad}_y(\gamma - \tilde{\gamma})(x,y)\|$, i.e., of bounding the derivatives of the approximation error.

3.4.2 Stability of Derivatives

In the case of polynomial interpolation, we can use an indirect approach: we construct a polynomial f^* of order $m-1$ which approximates f'. Its antiderivative $\tilde{f}$ will be a polynomial of order m with $f^* = \tilde{f}'$, therefore the derivative of $\tilde{f}$ will be a good approximation of f'. With a suitable generalization of the stability bound (3.25), we can then proceed as in Lemma 3.8 in order to prove that the interpolant of f is as least as good as $\tilde{f}$.

Our goal is therefore to establish a version of (3.25) relating to the derivatives of f and $\tilde{f}$. The key component of the proof is the following inverse estimate in the space of polynomials:

Lemma 3.13 (Markov's inequality iterated) *Let $\ell \in \mathbb{N}_0$, and let $u \in \mathcal{P}_m$. We have*

$$\|u^{(\ell)}\|_{\infty,[-1,1]} \leq \begin{cases} \left(\frac{m!}{(m-\ell)!} \right)^2 \|u\|_{\infty,[-1,1]} & \text{if } \ell \leq m, \\ 0 & \text{otherwise.} \end{cases} \tag{3.42}$$

Proof: By induction over ℓ. For $\ell = 0$, this inequality is trivial.

Let now $\ell \in \mathbb{N}_0$ be such that (3.42) holds. We have to prove the inequality for $\ell + 1$. If $\ell \geq m$, we have $\ell + 1 > m$ and $u^{(\ell+1)} = 0$, therefore (3.42) is trivial.

Otherwise, we have $u^{(\ell)} \in \mathcal{P}_{m-\ell}$ and can apply Markov's inequality [19, Theorem 4.1.4] in order to get

$$\|u^{(\ell+1)}\|_{\infty,[-1,1]} = \|(u^{(\ell)})'\|_{\infty,[-1,1]} \leq (m-\ell)^2 \|u^{(\ell)}\|_{\infty,[-1,1]}.$$

The induction assumption yields

$$\|u^{(\ell+1)}\|_{\infty,[-1,1]} \le (m-\ell)^2 \|u^{(\ell)}\|_{\infty,[-1,1]} \le (m-\ell)^2 \left(\frac{m!}{(m-\ell)!}\right)^2 \|u\|_{\infty,[-1,1]}$$
$$= \left(\frac{m!}{(m-\ell-1)!}\right)^2 \|u\|_{\infty,[-1,1]},$$

which is the desired estimate. ∎

Using this estimate, we can reduce the task of bounding the derivative of the interpolant to the task of bounding the interpolant itself, i.e., to the standard stability estimate (3.25).

Lemma 3.14 (Stability of derivatives) *Let $\ell \in \{0,\ldots,m\}$, let $f \in C^\ell[a,b]$. If $\mathfrak{I}_m^{[a,b]}$ satisfies the stability condition (3.25), it also satisfies the condition*

$$\|(\mathfrak{I}_m^{[a,b]}[f])^{(\ell)}\|_{\infty,[a,b]} \le \Lambda_m^{(\ell)} \|f^{(\ell)}\|_{\infty,[a,b]}$$

with the stability constant

$$\Lambda_m^{(\ell)} := \frac{\Lambda_m}{\ell!} \left(\frac{m!}{(m-\ell)!}\right)^2.$$

Proof: Let $\hat{f} := f \circ \Phi_{[a,b]} \in C^\ell[-1,1]$. We denote the Taylor expansion of the function $\hat{f}$ around zero by

$$\tilde{f}(x) := \sum_{k=0}^{\ell-1} \hat{f}^{(k)}(0) \frac{x^k}{k!}.$$

Since $\tilde{f}^{(\ell)} = 0$ and $\ell \le m$, we have

$$(\mathfrak{I}_m[\tilde{f}])^{(\ell)} = \tilde{f}^{(\ell)} = 0$$

and find

$$(\mathfrak{I}_m[\hat{f}])^{(\ell)} = (\mathfrak{I}_m[\hat{f}-\tilde{f}])^{(\ell)}.$$

We apply Lemma 3.13 to $u := \mathfrak{I}_m[\hat{f}-\tilde{f}] \in \mathcal{P}_m$ in order to get

$$\|(\mathfrak{I}_m[\hat{f}-\tilde{f}])^{(\ell)}\|_{\infty,[-1,1]} \le \left(\frac{m!}{(m-\ell)!}\right)^2 \|\mathfrak{I}_m[\hat{f}-\tilde{f}]\|_{\infty,[-1,1]} \le \Lambda_m \left(\frac{m!}{(m-\ell)!}\right)^2 \|\hat{f}-\tilde{f}\|_{\infty,[-1,1]}.$$

The standard error estimate for the Taylor expansion yields

$$\|\hat{f}-\tilde{f}\|_{\infty,[-1,1]} \le \frac{\|\hat{f}^{(\ell)}\|_{\infty,[-1,1]}}{\ell!},$$

and we conclude

$$\|(\mathfrak{I}_m[\hat{f}])^{(\ell)}\|_{\infty,[-1,1]} \le \frac{\Lambda_m}{\ell!} \left(\frac{m!}{(m-\ell)!}\right)^2 \|\hat{f}^{(\ell)}\|_{\infty,[-1,1]} = \Lambda_m^\ell \|\hat{f}^{(\ell)}\|_{\infty,[-1,1]}.$$

Let us now return our attention to the original function f on the interval $[a,b]$. Due to (3.22), we have

$$(\mathfrak{I}_m^{[a,b]}[f])^{(\ell)} = (\mathfrak{I}_m[\hat{f}] \circ \Phi_{[a,b]}^{-1})^{(\ell)} = \left(\frac{2}{b-a}\right)^\ell (\mathfrak{I}_m[\hat{f}])^{(\ell)} \circ \Phi_{[a,b]}^{-1},$$

which implies

$$\|(\mathfrak{I}_m^{[a,b]}[f])^{(\ell)}\|_{\infty,[a,b]} = \left(\frac{2}{b-a}\right)^\ell \|(\mathfrak{I}_m[\hat{f}])^{(\ell)}\|_{\infty,[-1,1]} \le \Lambda_m^{(\ell)} \left(\frac{2}{b-a}\right)^\ell \|\hat{f}^{(\ell)}\|_{\infty,[-1,1]}$$
$$= \Lambda_m^{(\ell)} \left(\frac{2}{b-a}\right)^\ell \|(f \circ \Phi_{[a,b]})^{(\ell)}\|_{\infty,[-1,1]} = \Lambda_m^{(\ell)} \left(\frac{2}{b-a}\right)^\ell \left(\frac{b-a}{2}\right)^\ell \|f^{(\ell)}\|_{\infty,[a,b]}$$
$$= \Lambda_m^{(\ell)} \|f^{(\ell)}\|_{\infty,[a,b]},$$

and this is the desired result. ∎

3.4.3 Construction of Approximants

We can use the stability estimate of Lemma 3.14 in order to prove a best-approximation inequality similar to the one from Lemma 3.8.

Theorem 3.15 (Approximation of derivatives) *Let $\ell \in \{0,\dots,m\}$, let $n \in \{0,\dots,m-\ell\}$ and let $f \in C^{m+1}[a,b]$. If (3.25) and (3.29) hold, we have*

$$\|f^{(\ell)} - (\mathfrak{I}_m^{[a,b]}[f])^{(\ell)}\|_{\infty,[a,b]} \leq \frac{2(\Lambda_m^{(\ell)}+1)}{(n+1)!}\left(\frac{b-a}{4}\right)^{n+1}\|f^{(\ell+n+1)}\|_{\infty,[a,b]}.$$

Proof: Let $f_\ell := f^{(\ell)}$, and let Let $\tilde{f}_\ell := \mathfrak{I}_n^{[a,b]}[f_\ell] \in \mathcal{P}_n \subseteq \mathcal{P}_{m-\ell}$. According to (3.31), we have

$$\begin{aligned}\|f_\ell - \tilde{f}_\ell\|_{\infty,[a,b]} &\leq \frac{2}{(n+1)!}\left(\frac{b-a}{4}\right)^{n+1}\|f_\ell^{(n+1)}\|_{\infty,[a,b]}\\ &= \frac{2}{(n+1)!}\left(\frac{b-a}{4}\right)^{n+1}\|f^{(\ell+n+1)}\|_{\infty,[a,b]}.\end{aligned}$$

For all $k \in \{0,\dots,\ell-1\}$, we construct $\tilde{f}_k \in \mathcal{P}_{m-k}$ inductively by

$$\tilde{f}_k(t) := \int_a^t \tilde{f}_{k-1}(s)\,\mathrm{d}s \qquad \text{for all } t \in [a,b].$$

This definition implies $\tilde{f}_k' = \tilde{f}_{k-1}$, and by induction $\tilde{f}_0^{(\ell)} = \tilde{f}_\ell$, i.e.,

$$\|(f-\tilde{f}_0)^{(\ell)}\|_{\infty,[a,b]} = \|f_\ell - \tilde{f}_\ell\|_{\infty,[a,b]} \leq \frac{2}{(n+1)!}\left(\frac{b-a}{4}\right)^{n+1}\|f^{(\ell+n+1)}\|_{\infty,[a,b]}.$$

Due to $\tilde{f}_0 \in \mathcal{P}_m$, we have

$$\begin{aligned}\|(f-\mathfrak{I}_m^{[a,b]}[f])^{(\ell)}\|_{\infty,[a,b]} &= \|(f-\tilde{f}_0+\mathfrak{I}_m^{[a,b]}[\tilde{f}_0]-\mathfrak{I}_m^{[a,b]}[f])^{(\ell)}\|_{\infty,[a,b]}\\ &\leq \|(f-\tilde{f}_0)^{(\ell)}\|_{\infty,[a,b]} + \|(\mathfrak{I}_m^{[a,b]}[f-\tilde{f}_0])^{(\ell)}\|_{\infty,[a,b]} \qquad (3.43)\end{aligned}$$

and can apply Lemma 3.14 to get

$$\|(\mathfrak{I}_m^{[a,b]}[f-\tilde{f}_0])^{(\ell)}\|_{\infty,[a,b]} \leq \Lambda_m^{(\ell)}\|(f-\tilde{f}_0)^{(\ell)}\|_{\infty,[a,b]}.$$

Combining this estimate with (3.43) yields

$$\|(f-\mathfrak{I}_m^{[a,b]}[f])^{(\ell)}\|_{\infty,[a,b]} \leq (1+\Lambda_m^{(\ell)})\|(f-\tilde{f}_0)^{(\ell)}\|_{\infty,[a,b]} \leq \frac{2(1+\Lambda_m^{(\ell)})}{(n+1)!}\left(\frac{b-a}{4}\right)^{n+1}\|f^{(\ell+n+1)}\|_{\infty,[a,b]},$$

which was to be proven. ∎

Combining this estimate with the technique used in the proof of Theorem 3.10 allows us to find an error estimate for the multi-dimensional interpolation operator.

We first consider the coordinate-wise operators $\mathfrak{I}_{m,i}^Q$:

Lemma 3.16 *Let $i,j \in \{1,\dots,d\}$. Let $\ell \in \{0,\dots,m\}$ and $n \in \{0,\dots,m-\ell\}$. We have*

$$\|\partial_j^\ell(\mathfrak{I}_{m,i}^Q[f])\|_{\infty,Q} \leq \begin{cases}\Lambda_m^{(\ell)}\|\partial_j f\|_{\infty,Q} & \text{if } i=j,\\ \Lambda_m\|\partial_j f\|_{\infty,Q} & \text{otherwise}\end{cases} \qquad \text{for all } f \in C^\ell(Q),$$

$$\|\partial_j^\ell f - \partial_j^\ell(\mathfrak{I}_{m,i}^Q[f])\|_{\infty,Q} \leq \begin{cases}\frac{2(\Lambda_m^{(\ell)}+1)}{(n+1)!}\left(\frac{b-a}{4}\right)^{n+1}\|\partial_i^{\ell+n+1}f\|_{\infty,Q} & \text{if } i=j,\\ \frac{2}{(n+1)!}\left(\frac{b-a}{4}\right)^{n+1}\|\partial_j^\ell\partial_i^{n+1}f\|_{\infty,Q} & \text{otherwise}\end{cases} \qquad \text{for all } f \in C^{\ell+n+1}(Q).$$

Proof: The case $i \neq j$ is simple: according to (3.33), we have

$$\partial_j^\ell(\mathfrak{I}_{m,i}^Q[f])(x) = \mathfrak{I}_{m,i}^Q[\partial_j^\ell f](x) \qquad \text{for all } x \in Q, f \in C^\ell(Q),$$
$$\partial_j^\ell(f - \mathfrak{I}_{m,i}^Q[f])(x) = \partial_j^\ell f(x) - \mathfrak{I}_{m,i}^Q[\partial_j^\ell f](x) \qquad \text{for all } x \in Q, f \in C^\ell(Q),$$

and applying Lemma 3.9 yields the desired result.

Let us now consider the case $i = j$. As in the proof of Lemma 3.9, we let $y_k \in [a_k, b_k]$ for all $k \in \{1, \ldots, i-1, i+1, \ldots, d\}$ and define

$$f_i : [a_i, b_i] \to \mathbb{R}, \qquad x \mapsto f(y_1, \ldots, y_{i-1}, x, y_{i+1}, \ldots, y_d)$$

for all $f \in C(Q)$. If $f \in C^\ell(Q)$ holds, we have

$$\partial_i^\ell f(y_1, \ldots, y_{i-1}, x, y_{i+1}, \ldots, y_d) = f_i^{(\ell)}(x),$$
$$\partial_i^\ell(\mathfrak{I}_{m,i}^Q[f])(y_1, \ldots, y_{i-1}, x, y_{i+1}, \ldots, y_d) = (\mathfrak{I}_m^{[a_i,b_i]}[f_i])^{(\ell)}(x)$$

for all $x \in [a_i, b_i]$. Lemma 3.14 yields the stability estimate, while we can use Theorem 3.15 in order to get

$$|\partial_i^\ell f(y_1, \ldots, y_{i-1}, x, y_{i+1}, \ldots, y_d) - \partial_i^\ell(\mathfrak{I}_{m,i}^Q[f])(y_1, \ldots, y_{i-1}, x, y_{i+1}, \ldots, y_d)| = |f_i^{(\ell)}(x) - (\mathfrak{I}_m^{[a_i,b_i]}[f_i])^{(\ell)}(x)|$$
$$\leq \frac{2(\Lambda_m^{(\ell)} + 1)}{(n+1)!} \left(\frac{b-a}{4}\right)^{n+1} \|f_i^{(\ell+n+1)}\|_{\infty,[a_i,b_i]} \leq \frac{2(\Lambda_m^{(\ell)} + 1)}{(n+1)!} \left(\frac{b-a}{4}\right)^{n+1} \|\partial_i^{\ell+n+1} f\|_{\infty,Q}$$

for all $x \in [a_i, b_i]$. ∎

Theorem 3.17 (Tensor-product interpolation) *Let $Q = [a_1, b_1] \times \ldots \times [a_d, b_d]$, let $m \in \mathbb{N}$, and let $\mu \in \mathbb{N}_0^d$ with $\mu_i \leq m$ for all $i \in \{1, \ldots, d\}$. Let $n \in \mathbb{N}_0$ with $n \leq m - \mu_i$ for all $i \in \{1, \ldots, d\}$. Let $\mathfrak{I}_m$ satisfy the stability estimate (3.25) and the error estimate (3.29). Let $\mathfrak{I}_m^Q$ be the m-th order tensor-product interpolation operator defined by (3.32). We let*

$$\Lambda_m^{(\mu)} := \prod_{i=1}^d (\Lambda_m^{(\mu_i)} + 1).$$

The error estimate

$$\|\partial^\mu(f - \mathfrak{I}_m^Q[f])\|_{\infty,Q} \leq \frac{2\Lambda_m^{(\mu)}}{(n+1)!} \sum_{i=1}^d \left(\frac{b_i - a_i}{4}\right)^{n+1} \|\partial_i^{n+1}\partial^\mu f\|_{\infty,Q}$$

holds for all $f \in C^{m+1}(Q)$.

Proof: We proceed as in the proof of Theorem 3.10: for all $i \in \{0, \ldots, d\}$, we introduce

$$P_i := \prod_{k=1}^i \mathfrak{I}_{m,k}^Q$$

and note $P_0 = I$ and $P_d = \mathfrak{I}_m^Q$. Let $f \in C^{m+1}(Q)$ and $i \in \{1, \ldots, d\}$. According to Lemma 3.16, we have

$$\|\partial_j^{(\mu_j)}(P_i[f])\|_{\infty,Q} = \|\partial_j^{(\mu_j)}(\mathfrak{I}_{m,i}^Q[P_{i-1}[f]])\|_{\infty,Q} \leq \|\partial_j^{(\mu_j)}(P_{i-1}[f])\|_{\infty,Q} \begin{cases} \Lambda_m^{(\mu_j)} & \text{if } i = j, \\ \Lambda_m & \text{otherwise,} \end{cases}$$

and due to $1 \leq \Lambda_m \leq \Lambda_m^{(\mu_j)} \leq \Lambda_m^{(\mu_j)} + 1$, a simple induction yields

$$\|\partial^\mu(P_i[f])\|_{\infty,Q} \leq \left(\prod_{j=1}^{i-1} (\Lambda_m^{(\mu_j)} + 1)\right) \|\partial^\mu f\|_{\infty,Q}$$

for all $i \in \{0, \ldots, d-1\}$.

The second inequality of Lemma 3.16 implies

$$\|\partial^\mu f - \partial^\mu(\mathfrak{I}^Q_{m,i}[f])\|_{\infty,Q} \leq \frac{2(\Lambda_m^{(\mu_i)}+1)}{(n+1)!}\left(\frac{b_i-a_i}{4}\right)^{n+1}\|\partial_i^{m+1}\partial^\mu f\|_{\infty,Q},$$

and proceeding as in the proof of Theorem 3.10 yields

$$\begin{aligned}\|\partial^\mu f - \partial^\mu(\mathfrak{I}^Q_m[f])\|_{\infty,Q} &\leq \sum_{i=1}^d \left(\prod_{j=1}^{i-1}(\Lambda_m^{(\mu_j)}+1)\right)\|\partial_j f - \partial_f(\mathfrak{I}^Q_{m,i}[f])\|_{\infty,Q}\\ &\leq \frac{2\Lambda_m^{(\mu)}}{(n+1)!}\sum_{i=1}^d\left(\frac{b_i-a_i}{4}\right)^{n+1}\|\partial_i^{n+1}\partial^\mu f\|_{\infty,Q},\end{aligned}$$

which is the inequality we had to prove. ∎

3.4.4 Application to the Generator Function

We assume that Q_t and Q_s are bounding boxes satisfying the admissibility condition (2.4) and require that the generator function γ is asymptotically smooth, i.e., satisfies

$$|\partial_x^\alpha\partial_y^\beta\gamma(x,y)| \leq C\,|\alpha+\beta|!\;c_0^{|\alpha+\beta|}\,\|x-y\|^{-|\alpha+\beta|-\sigma}$$

for all $x \in Q_t$, $y \in Q_s$ and all multi-indices $\alpha, \beta \in \mathbb{N}_0^d$.

We assume that $g = \partial_x^\mu\partial_y^\zeta\gamma$ holds for $\mu, \zeta \in \mathbb{N}_0^d$.

We will consider only the case $\operatorname{diam}(Q_t) \leq \operatorname{diam}(Q_s)$, i.e., we will approximate the function γ_x defined by

$$\gamma_x : Q_t \to C^\infty(Q_s), \qquad x \mapsto \gamma(x,\cdot),$$

by an interpolant $\mathfrak{I}_m^{Q_t} g_x$. The case $\operatorname{diam}(Q_t) > \operatorname{diam}(Q_s)$, in which we approximate a similarly-defined γ_y, can be handled similarly.

We assume that the interpolation order $m \in \mathbb{N}$ is sufficiently high, i.e., that $\mu_i \leq m$ holds for all $i \in \{1, \ldots, d\}$, and we fix $n \in \mathbb{N}_0$ with $n \leq m - \mu_i$ for all $i \in \{1, \ldots, d\}$.

For $\partial^\mu\gamma_x$, we have

$$\|\partial^\alpha\partial^\mu g_x(x)\|_{\infty,Q_t} \leq C\,|\alpha+\mu|!\;c_0^{|\alpha+\mu|}\operatorname{dist}(Q_t,Q_s)^{-|\alpha|-(\sigma+|\mu|)}.$$

Using this estimate, we can derive the bound

$$\|\partial_i^{n+1}\partial_\mu g_x\|_{\infty,Q_t} \leq C\,(|\mu|+n+1)!\;c_0^{|\mu|+n+1}\operatorname{dist}(Q_t,Q_s)^{-(n+1)-(\sigma+|\mu|)}$$

for all $i, j \in \{1, \ldots, d\}$, and combining this bound with Theorem 3.17 yields

$$\|\partial^\mu\gamma - \partial^\mu(\mathfrak{I}^Q_m[\gamma])\|_{\infty,Q} \leq \frac{2C\Lambda_m^{(\mu)}c_0^{|\mu|}}{\operatorname{dist}(Q_t,Q_s)^{\sigma+|\mu|}}\frac{(|\mu|+n+1)!}{(n+1)!}\sum_{i=1}^d\left(\frac{(b_i-a_i)c_0}{4\operatorname{dist}(Q_t,Q_s)}\right)^{n+1}.$$

We have $b_i - a_i \leq \operatorname{diam}(Q_t)$, and since Q_t and Q_s are admissible, we also find

$$\operatorname{dist}(Q_t,Q_s)^{-1} \leq \eta\operatorname{diam}(Q_t)^{-1},$$

and combining both estimates yields

$$\frac{(b_i-a_i)c_0}{4\operatorname{dist}(Q_t,Q_s)} \leq \frac{c_0\eta}{4}.$$

As in the standard case, we assume $\eta < 4/c_0$, i.e., $c_0\eta/4 < 1$, and conclude

$$\|\partial^\mu\gamma - \partial^\mu(\mathfrak{I}_m^Q[\gamma])\|_{\infty,Q} \leq \frac{2Cd\Lambda_m^{(\mu)}c_0^{|\mu|}}{\operatorname{dist}(Q_t,Q_s)^{\sigma+|\mu|}}\frac{(|\mu|+n+1)!}{(n+1)!}\left(\frac{c_0\eta}{4}\right)^n, \tag{3.44}$$

so the derivative of the interpolant of γ will converge to the derivative of γ with the same rate as in the standard case.

We introduce the polynomial

$$C_{\mathrm{in}}^{(\mu)}(n) := 2Cd\left(\prod_{i=1}^d \frac{m+1}{\mu_i!}\left(\frac{m!}{(m-\mu_i)!}\right)^2\right)c_0^{|\mu|}\frac{(|\mu|+n+1)!}{(n+1)!}$$

and find that (3.44) implies

$$|g(x,y)-\tilde{g}(x,y)| \leq \frac{C_{\mathrm{in}}^{(\mu)}(n)}{\operatorname{dist}(Q_t,Q_s)^{\sigma+|\mu|}}\left(\frac{c_0\eta}{4}\right)^{n+1}$$

for the separable approximation given by

$$\tilde{g}(x,y) = \sum_{\nu\in K}(\partial^\mu\mathcal{L}_\nu^t)(x)(\partial_y^\zeta)g(x_\nu^t,y).$$

This estimate is very similar to the estimate (3.41) we have found for the approximation without derivatives, only the polynomial $C_{\mathrm{in}}^{(\mu)}$ is of higher order and we only get an exponent of $n+1$ instead of $m+1$ in the rightmost term.

We can conclude that the derivatives of the interpolant will converge under exactly the same conditions as before and with the same asymptotic convergence rate, only the additional constants are higher.

3.5 Example: Boundary Element Method in 2D

In Example 2.11, we have considered the construction of a cluster tree for a curve in two-dimensional space. Now we will solve integral equations on this curve by the techniques introduced in this chapter.

We are interested in a *boundary integral problem*, i.e., the set Ω will be a submanifold. In our case, Ω is a one-dimensional submanifold of $\mathbb{R}^2$, i.e., a curve. Since we are interested in integral equations, we have to recall the meaning of an integral on a curve.

3.5.1 Curve Integrals

Let $\gamma : [0,1] \to \mathbb{R}^2$ be a continuously differentiable function that is injective in $[0,1[$. We denote its range by $\Gamma := \gamma([0,1])$. Let $u \in C(\Gamma)$. Similar to the construction of Riemann integrals, we introduce a partition $0 = x_0 < x_1 < \ldots < x_n = 1$ of the interval $[0,1]$ and consider the sum

$$I_x := \sum_{i=1}^n u(\gamma(x_i))\|\gamma(x_i)-\gamma(x_{i-1})\|.$$

Lemma 3.18 (Curve integral) *Let $\varepsilon \in \mathbb{R}_{>0}$. There is a $\delta \in \mathbb{R}_{>0}$ such that for all partitions $0 = x_0 < x_1 < \ldots < x_n = 1$ with $x_i - x_{i-1} < \delta$ ($i \in \{1,\ldots,n\}$) we have*

$$\left|I_x - \int_0^1 u(\gamma(y))\|\gamma'(y)\|\,\mathrm{d}y\right| \leq \varepsilon.$$

Proof: Since $u \circ \gamma$ and $\|\gamma'\|$ are continuous on the compact set $[0,1]$, they are uniformly continuous. Let $\hat{\varepsilon} := \sqrt{\varepsilon + a^2/4} - a/2$ for

$$a := \int_0^1 |u(\gamma(y))| + \|\gamma'(y)\| \, \mathrm{d}y.$$

Due to uniform continuity, there is a $\delta \in \mathbb{R}_{>0}$ such that

$$|u \circ \gamma(x) - u \circ \gamma(y)| < \hat{\varepsilon} \quad \text{and} \quad |\|\gamma'(x)\| - \|\gamma'(y)\|| < \hat{\varepsilon}$$

holds for all $x, y \in [0,1]$ with $|x - y| < \delta$.

Let $0 = x_0 < x_1 < \ldots < x_n = 1$ with $x_i - x_{i-1} < \delta$ for all $i \in \{1, \ldots, n\}$. By the differential mean value theorem, we find an $\hat{x}_i \in [x_{i-1}, x_i]$ with

$$\gamma'(\hat{x}_i) = \frac{\gamma(x_i) - \gamma(x_{i-1})}{x_i - x_{i-1}}.$$

This implies

$$\begin{aligned}
\left| I_x - \int_0^1 u(\gamma(y)) \|\gamma'(y)\| \, \mathrm{d}y \right| &= \left| \sum_{i=1}^n \int_{x_{i-1}}^{x_i} \left(u(\gamma(x_i)) \left\| \frac{\gamma(x_i) - \gamma(x_{i-1})}{x_i - x_{i-1}} \right\| - u(\gamma(y)) \|\gamma'(y)\| \right) \mathrm{d}y \right| \\
&\leq \sum_{i=1}^n \left| \int_{x_{i-1}}^{x_i} \left(u(\gamma(x_i)) \|\gamma'(\hat{x}_i)\| - u(\gamma(y)) \|\gamma'(y)\| \right) \mathrm{d}y \right| \\
&\leq \sum_{i=1}^n \int_{x_{i-1}}^{x_i} \left(|u(\gamma(x_i))| \; |\|\gamma'(\hat{x}_i)\| - \|\gamma'(y)\|| + |u(\gamma(x_i)) - u(\gamma(y))| \|\gamma'(y)\| \right) \mathrm{d}y \\
&\leq \sum_{i=1}^n \int_{x_{i-1}}^{x_i} \left(|u(\gamma(x_i))| \varepsilon + \varepsilon \|\gamma'(y)\| \right) \mathrm{d}y \\
&\leq \sum_{i=1}^n \int_{x_{i-1}}^{x_i} \left(\hat{\varepsilon}^2 + |u(\gamma(y))| \hat{\varepsilon} + \hat{\varepsilon} \|\gamma'(y)\| \right) \mathrm{d}y \\
&= \hat{\varepsilon}^2 + \hat{\varepsilon} \int_0^1 \left(|u \circ \gamma(y)| + \|\gamma'(y)\| \right) \mathrm{d}y = \hat{\varepsilon}^2 + \hat{\varepsilon} a = \varepsilon.
\end{aligned}$$

■

Now we can define the curve integral: let $(\gamma_i)_{i=1}^m$ be a tuple of injective functions in $C^1([0,1], \mathbb{R}^2)$. For all $i \in \{1, \ldots, m\}$, we set $\Gamma_i := \gamma_i([0,1])$. The *curve integral* over the piecewise differentiable curve $\Gamma := \bigcup_{i=1}^m \Gamma_i$ is given by

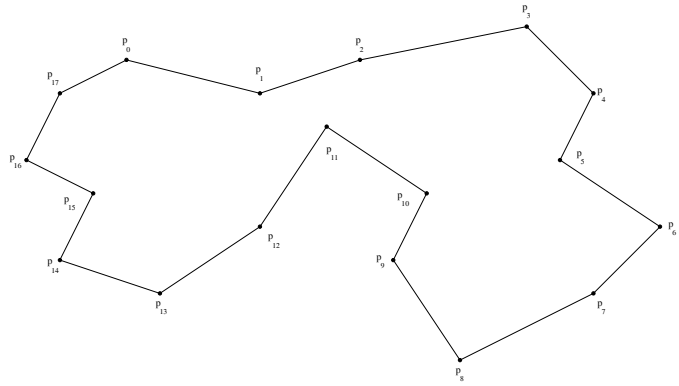
$$\int_\Gamma u(x) dx := \sum_{i=1}^m \int_0^1 u(\gamma_i(y)) \|\gamma_i'(y)\| \, \mathrm{d}y.$$

3.5.2 Single Layer Potential Operator

We fix n points $p_0, \ldots, p_{n-1} \in \mathbb{R}^2$, set $p_n := p_0$ and define the affine parametrisations

$$\gamma_i : [0,1] \to \mathbb{R}^2, \quad y \mapsto p_{i-1}(1-y) + p_i y,$$

for $i \in \{1, \ldots, n\}$. As long as $p_i \neq p_j$ holds for all $i, j \in \{0, \ldots, n-1\}$ with $i \neq j$, this defines a polygonal curve $\Gamma := \bigcup_{i=1}^m \gamma_i([0,1])$.



On the curve Γ, we can now define the *single layer potential* operator

$$\mathcal{G}_{\mathrm{slp}}[u](x) := -\frac{1}{2\pi} \int_\Gamma \log(\|x-y\|) u(y) \,\mathrm{d}y$$

and the corresponding bilinear form

$$a_{\mathrm{slp}}(u,v) := -\frac{1}{2\pi} \int_\Gamma v(x) \int_\Gamma \log(\|x-y\|) u(y) \,\mathrm{d}y \,\mathrm{d}x.$$

We discretise $a_{\mathrm{slp}}(\cdot,\cdot)$ by piecewise constant functions $(\varphi_i)_{i=1}^n$ defined through

$$\varphi_i \circ \gamma_j \equiv \delta_{ij}$$

for $i,j \in \mathcal{I} := \{1,\ldots,n\}$. The coefficients of the corresponding matrix are given by

$$\begin{aligned} G_{ij} &= a_{\mathrm{slp}}(\varphi_j,\varphi_i) = -\frac{1}{2\pi} \int_\Gamma \varphi_i(x) \int_\Gamma \log(\|x-y\|) \varphi_j(y) \,\mathrm{d}y \,\mathrm{d}x \\ &= -\frac{1}{2\pi} \|p_i - p_{i-1}\| \, \|p_j - p_{j-1}\| \int_0^1 \int_0^1 \log(\|\gamma_i(x) - \gamma_j(y)\|) \,\mathrm{d}y \,\mathrm{d}x. \end{aligned}$$

Now we are in the situation of Section 3.2 and can construct a hierarchical matrix by replacing the logarithmic kernel $g(x,y) := \log(\|x-y\|)$ by degenerate approximations.

3.5.3 Implementation

We have already considered the construction of suitable cluster trees and admissible partitions in Examples 2.11 and 2.20, so we will focus on the problem of computing the entries of the hierarchical matrix represented by a `supermatrix` in our code.

The treatment of the full matrices involves the efficient evaluation of singular integrals and is not the subject of our investigation. It suffices to say that we provide a function `integrate_nearfield` that initialises these matrices.

Let us now consider the treatment of the low-rank blocks. They correspond to admissible pairs (t,s) of clusters and require the evaluation of a degenerate approximation of the kernel function. We assume that $\operatorname{diam}(Q^t) \leq \operatorname{diam}(Q^s)$, so the approximation is given by

$$\tilde{g}(x,y) = \sum_{\nu \in K} \log(\|x_\nu^t - y\|) \mathcal{L}_\nu^t(x)$$

and we have to compute the matrices

$$A_{i\nu}^{t,s} = \int_\Gamma \varphi_i(x) \mathcal{L}_\nu^t(x) \,\mathrm{d}x = \|p_i - p_{i-1}\| \int_0^1 \mathcal{L}_\nu^t(\gamma_i(x)) \,\mathrm{d}x, \tag{3.45}$$

$$B_{j\nu}^{t,s} = -\frac{1}{2\pi}\int_\Gamma \varphi_j(y)\log(\|x_\nu^t - y\|)\,\mathrm{d}y = -\frac{1}{2\pi}\|p_j - p_{j-1}\|\int_0^1 \log(\|x_\nu^t - \gamma_j(y)\|)\,\mathrm{d}y. \tag{3.46}$$

Since γ_i is affine, the first integrand is a polynomial of degree m, so we can apply an exact quadrature rule for its evaluation.

In order to implement this computation, we need the following data:

- An array `vertex` of dimension `n` containing the coordinates of the points $(p_i)_{i=0}^{n-1}$.
- Arrays `xq` and `wq` of dimension `q` containing the points and weights of a suitable quadrature rule. For the typical choice, Gauss quadrature, we can use the library routine `build_gauss`.
- An array `l` of dimension `p` containing the transformed interpolation points. This array can be computed by the function given in Subsection 3.3.2.

Filling the matrix $B^{t,s}$ requires us to integrate the kernel function for points x_ν^t on intervals given by p_{i-1} and p_i. In our example, this can be done analytically (cf. Exercise 3). In more general situations, we can use the same quadrature rule as in the case of the integration of the Lagrange polynomials.

The `supermatrix` structure can be initialised by a simple recursion: If the `supermatrix` contains an `rkmatrix`, we compare the diameters of the clusters involved and use the procedure described above to initialise the fields `a` and `b` of the `rkmatrix`.

If the `supermatrix` contains a `fullmatrix`, we evaluate singular integrals and fill its field `e`.

Otherwise, we proceed recursively with the subblocks that are given by the array `s`.

3.6 Exercises

3.6.1 Practice

Exercise 3 *Let $\gamma : [0,1] \to \mathbb{R}^2$ and $c \in \mathbb{R}^2$ be given by*

$$\gamma(t) := \begin{pmatrix} s_x + td_x \\ s_y + td_y \end{pmatrix} \quad \text{and} \quad c := \begin{pmatrix} c_x \\ c_y \end{pmatrix}.$$

We assume that $c \notin \gamma([0,1])$. Write a C *function*

```
static double
collocation_logarithm(double sx, double sy, double dx, double dy,
                      double cx, double cy);
```

that computes the value of the integral

$$\int_0^1 \log(\|\gamma(t) - c\|_2)\ \|\gamma'(t)\|_2\,\mathrm{d}t.$$

There are two ways of solving this exercise: you can compute the value of the integral explicitly by using the equation

$$\int \log(a^2 + x^2)\,\mathrm{d}x = x\log(a^2 + x^2) - 2x + 2a\arctan\left(\frac{x}{a}\right),$$

or you can use Gauss quadrature. Gauss quadrature points can be computed by the routine `build_gauss` *in the file* `quadrature.c`.

Exercise 4 (BEM) *The structure* `curvebemfactory` *contains the data necessary for the discretisation of the single layer potential on a curve:*

- *An integer* `n` *and an array* `vertex` *containing the geometrical information of the curve, i.e., the points* $(p_i)_{i=0}^{p-1}$.
- *Arrays* `xq` *and* `wq` *containing* `q` *quadrature points.*
- *An array* `xp` *containing* `p` *interpolation points.*
- *An auxiliary array* `l` *that can be used to store transformed interpolation points.*

Use the given function `transform_interpolation_points` *to implement the functions*

```
static void
integrate_polynomial(pccluster t,
                     pcurvebemfactory bfactory,
                     double *X, int ldX);

static void
integrate_kernel(pccluster t, pccluster s,
                 pcurvebemfactory bfactory,
                 double *X, int ldX);
```

that initialise the matrices `X` *as follows: for the first function, the matrix* `X` *is filled with the entries of the matrix* $A^{t,s}$ *from* (3.45). *For the second function, it is filled with the entries of the matrix* $B^{t,s}$.

Chapter 4

Cross Approximation

In the previous chapter we have constructed a degenerate kernel approximation $\tilde{g}(x,y) \approx g(x,y)$. The subblock $R := \tilde{G}|_{\hat{t}\times\hat{s}}$ of the stiffness matrix using the degenerate kernel $\tilde{g}$ instead of g is of low rank and can be represented by an `rkmatrix` AB^T. The rank k in this representation is independent of the quadrature used to compute the matrix entries and it is independent of the discretisation scheme (Nystrøm, Collocations, Galerkin).

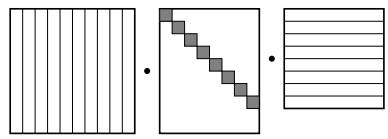
In this section we consider methods that start with the already discretised matrix block $M := G|_{\hat{t}\times\hat{s}}$ and compute a (low rank) approximation up to a prescribed error ε. For the more elaborate (and more heuristic) methods it is not necessary to assemble the whole matrix block in advance, the method will request to compute particular entries of the matrix.

4.1 Singular Value Decomposition

The minimal rank approximation R of $M \in \mathbb{R}^{\hat{t}\times\hat{s}}$ for a prescribed error bound $\|R-M\|_2 \le \varepsilon$ is given in the following Lemma.

Lemma 4.1 (Best approximation via SVD) *Let the singular value decomposition of $M \in \mathbb{R}^{\hat{t}\times\hat{s}}$ be given by*

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



with unitary matrices U and V (columns orthonormal) and diagonal (rectangular) matrix

$$\Sigma = \operatorname{diag}(\sigma_1,\ldots,\sigma_{\min(\#\hat{t},\#\hat{s})}), \quad \sigma_1 \ge \sigma_2 \ge \cdots \ge \sigma_{\min(\#\hat{t},\#\hat{s})} \ge 0.$$

Then the matrix

$$R := \sum_{\nu=1}^{k} u_\nu \sigma_\nu v_\nu^T, \qquad \sigma_k > \varepsilon \ge \sigma_{k+1},$$

is a minimal rank approximation of M that fulfils $\|M-R\|_2 \le \varepsilon$, i.e., $R = \operatorname{argmin}_{R\in\mathcal{R}(k)}\|M-R\|_2$.

Proof: [21, Theorem 2.5.3] ∎

The rank k of the minimal rank approximation R obviously depends on the desired accuracy ε. But it also depends on the perturbation of M due to an inexact quadrature.

Example 4.2 (Influence of quadrature errors) *We consider the model problem from Section 1.1 where the matrix block $G_{\hat{t}\times\hat{s}}$ corresponding to the subdomain $[0,\frac{1}{4}]\times[\frac{3}{4},1]$ can be approximated by low rank provided that the quadrature is exact. In practice this will almost never be the case (due to the complexity). Instead, one lowers the order of the quadrature formula in the farfield where the kernel is fairly smooth. In particular, one can replace farfield entries $G_{i,j}$ by zero if the distance between* $\operatorname{supp}\varphi_i$ *and* $\operatorname{supp}\varphi_j$ *is large compared to* $\operatorname{diam}(\operatorname{supp}\varphi_i\times\operatorname{supp}\varphi_j)$*. In our model problem we assume that the meshwidth h is small enough so that matrix entries $G_{i,j}$ can be replaced by zero if $|i-j|>\frac{3}{4}n$. For $i=0$ all entries $G_{0,j}$ with $j>\frac{3}{4}n$ are zero, for $i=1$ all entries $G_{1,j}$ with $j>\frac{3}{4}n+1$ are zero and so forth. The pattern of the matrix block $G_{\hat{t}\times\hat{s}}$ is that of a*

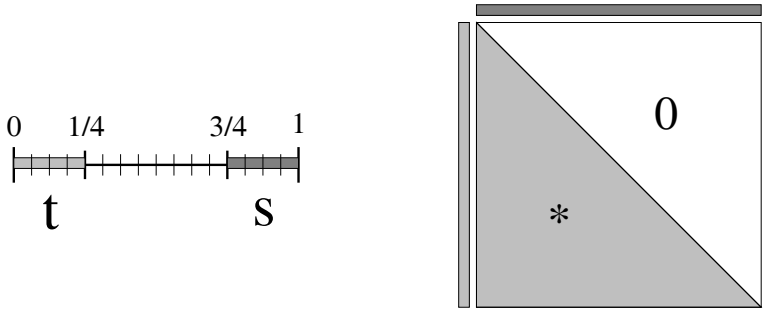


Figure 4.1: The matrix $G_{\hat{t}\times\hat{s}}$ corresponding to the subdomain $[0,\frac{1}{4}]\times[\frac{3}{4},1]$ is lower triangular.

lower triangular matrix (cf. Figure 4.1). This means that for an accuracy $\varepsilon \ll \min\{|G_{i,j}|\neq 0 \mid i\in\hat{t}, j\in\hat{s}\}$ any approximation $\tilde{G}|_{\hat{t}\times\hat{s}}$ of $G|_{\hat{t}\times\hat{s}}$ with $\|\tilde{G}|_{\hat{t}\times\hat{s}}-G|_{\hat{t}\times\hat{s}}\|_2\leq\varepsilon$ has to be of full rank.

The previous example is *not* an academic or practically unimportant example. In fact, whenever the quadrature order is changed within a matrix block, we expect the approximation rank to be $k=\mathcal{O}(n)$ as the desired accuracy ε reaches the quadrature error. If we forbid a change in the quadrature error within a single matrix block (in order to get rid of this side effect) and instead use the highest quadrature, proposed for an entry (i,j) in the respective block, for all entries in the block, then the computational effort is much higher, as can be seen in Figure 4.2.

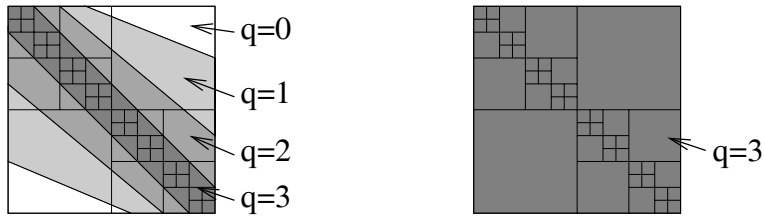


Figure 4.2: The matrix G from the model problem of Section 1.1 is assembled with different quadrature orders $q\in\{0,1,2,3\}$ in different parts of the matrix. The highest order is used close to the diagonal. If we forbid a change in the quadrature order within a single block, then *all* blocks have to be assembled with the highest order $q=3$.

Conclusion: When applying rank revealing decompositions, then the accuracy of the quadrature has to be better than the desired accuracy of the low rank approximation.

4.2 Cross Approximation

An interesting approach for a special low rank approximation is proposed in [26] under the name *skeleton approximation* or *cross approximation*. Let us fix a block $\hat{t}\times\hat{s}\subset\mathcal{I}\times\mathcal{I}$ and a matrix block $M\in\mathbb{R}^{\hat{t}\times\hat{s}}$. The

idea of cross approximation is to choose (in a clever way that we describe later) a small set $s^* \subset \hat{s}$ of pivot columns and a small set $t^* \subset \hat{t}$ of pivot rows so that for a matrix $S \in \mathbb{R}^{s^* \times t^*}$ there holds (cf. Figure 4.3)

$$\|M - \tilde{M}\|_2 \leq \varepsilon, \qquad \tilde{M} := M|_{\hat{t} \times s^*} \cdot S \cdot M|_{t^* \times \hat{s}} \in \mathcal{R}(\min\{\#s^*, \#t^*\}).$$

Since only $\#t^*$ rows and $\#s^*$ columns of the matrix M are used in the representation of $\tilde{M}$, this is an

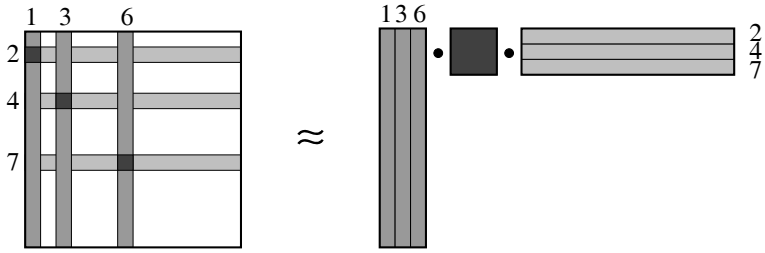


Figure 4.3: The matrix M is approximated by a combination of few rows $t^* = \{2, 4, 7\}$ and columns $s^* = \{1, 3, 6\}$ of the matrix. A diagonal entry $(i, j) \in t^* \times s^*$ defines the (rank 1) cross $(M|_{\{i\} \times \hat{s}}) \frac{1}{M_{i,j}} (M|_{\{j\} \times \hat{t}})$.

efficient way to assemble the matrix $\tilde{M}$ in `rkmatrix` format with rank $k = \min\{\#t^*, \#s^*\}$.

There is a positive result proven in [26]: if we assume that there exists a sufficiently good low rank approximation, then there also exists a cross approximation with almost the same approximation quality.

Theorem 4.3 (Existence of cross approximations) *Let $M, R \in \mathbb{R}^{\hat{t} \times \hat{s}}$ be matrices with $\|M - R\| \leq \varepsilon$ and $\operatorname{rank}(R) \leq k$. Then there exists a subset $s^* \subset \hat{s}$ of pivot columns and a subset $t^* \subset \hat{t}$ of pivot rows and a matrix $S \in \mathbb{R}^{t^* \times s^*}$ with*

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

Proof: [26] ■

In the following we introduce two constructive heuristics for the assembly of a cross approximation. Both algorithms compute successive rank one approximations. The first one is of quadratic complexity, the second one of linear complexity.

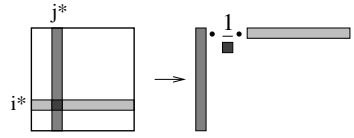
4.3 Cross Approximation with Full Pivoting

Let $M \in \mathbb{R}^{\hat{t} \times \hat{s}} \neq 0$ be given. All entries of M have to be computed in advance.

An $\mathcal{R}(1)$-approximation of M using *full pivoting* is constructed in two steps:

1. Determine the index pair (i^*, j^*) with maximal entry $|M_{i^*,j^*}|$ in modulus. Set $\delta := M_{i^*,j^*}$.
2. Compute the entries

$$a_i := M_{i,j^*}, \quad i \in \hat{t}, \qquad b_j := M_{i^*,j}/\delta, \quad j \in \hat{s}.$$



The matrix $R := ab^T \in \mathcal{R}(1)$ is the sought rank one approximation.

An $\mathcal{R}(k)$-approximation of M consists of k steps of the above procedure applied to the remainder $M - \sum_{\nu=1}^{\mu} a^\nu (b^\nu)^T$. The rank one approximation is denoted by R_1, the rank two approximation by R_2 and so forth. The complete cross approximation algorithm with full pivoting is given in Algorithm 2.

This approximation scheme is able to find the exact representation of a rank k matrix in k steps, i.e., $\sum_{\nu=1}^{k} a^\nu (b^\nu)^T = M$ if $\operatorname{rank}(M) = k$.

Algorithm 2 Cross approximation with full pivoting and quadratic complexity $\mathcal{O}(k\#\hat{t}\#\hat{s})$.

The input of the algorithm is the matrix $M \in \mathbb{R}^{\hat{t}\times\hat{s}}$ which is overwritten during the procedure. The output is a rank k approximation $\sum_{\nu=1}^{k} a^\nu (b^\nu)^T$ of M.

for $\nu = 1, \ldots, k$ **do**

Compute the maximal entry in modulus

$$(i_\nu, j_\nu) := \operatorname{argmax}_{(i,j)} |M_{i,j}|, \qquad \delta := M_{i_\nu, j_\nu}.$$

if $\delta = 0$ **then**

the algorithm terminates with the exact rank $\nu - 1$ representation $R_{\nu-1}$ of the input matrix M.

else

Compute the entries of the vectors a^ν, b^ν:

$$(a^\nu)_i := M_{i,j_\nu}, \quad i \in \hat{t}, \qquad (b^\nu)_j := M_{i_\nu,j}/\delta, \quad j \in \hat{s}.$$

Subtract the rank one approximation

$$M_{i,j} := M_{i,j} - (a^\nu)_i (b^\nu)_j, \qquad i \in \hat{t}, j \in \hat{s}.$$

end if

end for

Lemma 4.4 (Exact reproduction of rank k matrices) *Let M be a matrix of rank exactly k. Then the matrix $R_k := \sum_{\nu=1}^{k} a_\nu b_\nu^T$ given by Algorithm 2 equals M.*

Proof: We will prove that $M_{k'} := M - \sum_{\nu=1}^{k'} a^\nu (b^\nu)^T$ is of rank $k - k'$ for $k' \in \{0, \ldots, k\}$ by induction, where the start $k' = 0$ is trivial. Now let $M_{k'}$ be of rank $k - k'$. Let V denote the $k - k'$-dimensional image of $M_{k'}$ and $W = V^\perp$ the $n - k + k'$-dimensional complement in $\mathbb{R}^{n\times m}$. Let V' and W' denote the corresponding spaces for $M_{k'+1} = M_{k'} - a^{k'+1}(b^{k'+1})^T$. Since by construction (Algorithm 2) $a^{k'+1}$ is a column $j^* := j_{k'+1}$ of $M_{k'}$ it must belong to V. Therefore $V' \subset V$ and $W \subset W'$. Let $i^* := i_{k'+1}$ be the corresponding row pivot index. Then there holds

$$\begin{aligned} e_{i^*}^T M_{k'} &= (b^{k'+1})^T \delta \neq 0, \\ e_{i^*}^T (M_{k'} - a^{k'+1}(b^{k'+1})^T) &= (b^{k'+1})^T \delta - e_{i^*}^T a^{k'+1} (b^{k'+1})^T \;=\; (b^{k'+1})^T \delta - \delta (b^{k'+1})^T \;=\; 0, \end{aligned}$$

so that $e_{i^*} \in W' \setminus W$, i.e., the dimension of the image V' of $M_{k'+1}$ is at most $\dim(V) - 1 = k - k' - 1$. Since $M_{k'+1}$ is a rank one perturbation of $M_{k'}$, we conclude $\operatorname{rank}(M_{k'+1}) = \operatorname{rank}(M_{k'}) - 1 = k - k' - 1$.

For $k' = k$ we get $\operatorname{rank}(M_k) = 0$ so that $M = \sum_{\nu=1}^{k} a^\nu (b^\nu)^T$. ∎

Lemma 4.5 (Interpolation property) *Let M be a matrix of rank at least $k \geq 1$ and R_k the cross approximation from Algorithm 2. For any row pivot index i^* and any column pivot index j^* there holds*

$$R_k e_{j^*} = M_{\hat{t}\times\{j^*\}} \quad \text{and} \quad e_{i^*}^T R_k = M_{\{i^*\}\times\hat{s}},$$

i.e., R_k exactly reproduces the pivot columns and rows of M.

Proof: The statement holds for $k = 1$:

$$(R_1 e_{j^*})_i = a_i^1 b_{j^*}^1 = M_{i,j^*}, \quad (e_{i^*}^T R_1)_j = a_{i^*}^1 b_j^1 = M_{i^*,j}$$

so that for the remainder M_1 the row i^* and column j^* is zero. We prove by induction over $k' = 1, \ldots, k$ that each $R_{k'}$ fulfils the statement for the first k' pivot elements, where the start of the induction is already proven.

The matrix $M_{k'}$, $1 \leq k' < k$ has zero columns and rows for the first k' pivot columns and rows. For the $k'+1$-st column pivot element j^* there holds

$$(R_{k'+1}e_{j^*})_i = (R_{k'})_{i,j^*} + a_i^{k'+1}b_{j^*}^{k'+1}(R_{k'})_{i,j^*} + (M_{k'})_{i,j^*} = M_{i,j^*}$$

and analogously for i^*. For any of the first k' pivot elements j^* we conclude

$$(R_{k'+1}e_{j^*})_i = (R_{k'})_{i,j^*} + a_i^{k'+1}b_{j^*}^{k'+1}(R_{k'})_{i,j^*} + (M_{k'})_{i,j^*} = (R_{k'})_{i,j^*} + 0 = M_{i,j^*}.$$

■

Lemma 4.6 *The cross approximation of a matrix M of rank at least k by the matrix R_k from above is of the form*

$$R_k := \sum_{\nu=1}^{k} a^\nu (b^\nu)^T \quad = \quad M|_{\hat{t}\times\mathcal{P}_{\text{cols}}} \cdot (M|_{\mathcal{P}_{\text{rows}}\times\mathcal{P}_{\text{cols}}})^{-1} \cdot M|_{\mathcal{P}_{\text{rows}}\times\hat{s}} =: \widehat{R}_k,$$

where $\mathcal{P}_{\text{rows}}$ is the set of pivot rows and $\mathcal{P}_{\text{cols}}$ the respective set of pivot columns. Note that the ordering of the pivot rows and columns is irrelevant.

Proof: By construction both matrices R_k and $\widehat{R}_k$ are of rank k. Let $j_\ell \in \mathcal{P}_{\text{cols}}$ be the ℓ-th pivot column. Then for the j_ℓ-th unit vector e_{j_ℓ} there holds

$$\widehat{R}_k e_{j_\ell} = M|_{\hat{t}\times\mathcal{P}_{\text{cols}}} \cdot (M|_{\mathcal{P}_{\text{rows}}\times\mathcal{P}_{\text{cols}}})^{-1} \cdot M|_{\mathcal{P}_{\text{rows}}\times\{j_\ell\}} = M|_{\hat{t}\times\mathcal{P}_{\text{cols}}} e_\ell = M|_{\hat{t}\times\{j_\ell\}}.$$

According to Lemma 4.5 both R_k and $\widehat{R}_k$ are identical on the span of the unit vectors e_{j^*}, $j^* \in \mathcal{P}_{\text{col}}$. Since the image of them spans the whole image of R_k and $\widehat{R}_k$, both matrices are identical. ■

For matrices M with full rank but exponentially decaying singular values there is no guarantee for a good approximation quality of R_k. Even if we could prove such a result, the quadratic complexity $\mathcal{O}(k\#\hat{t}\#\hat{s})$ is not efficient enough for large scale problems. The reason for the inefficiency comes from the determination of the pivot indices (i^*, j^*).

4.4 Cross Approximation with Partial Pivoting

In each step $\nu = 1, \ldots, k$ of Algorithm 2 we have to

- determine the pivot pair (i^*, j^*) (complexity $\mathcal{O}(\#\hat{t}\#\hat{s})$)
- compute the two vectors a^ν, b^ν (complexity $\mathcal{O}(\#\hat{t} + \#\hat{s})$) and
- update the matrix M (complexity $\mathcal{O}(\#\hat{t}\#\hat{s})$).

The bottleneck is the determination of the pivot pairs and the update of the matrix M.

In the following we introduce a heuristic where the determination of the pivot pair (i^*, j^*) is changed so that we do not have to compute all entries of M in advance. Also, we omit the update of M and instead use the representation $M_\nu = M - \sum_{\nu=1}^{k'} a^\nu (b^\nu)^T$ from above. The only open question is: how do we choose the pivot indices (i^*, j^*) ?

The idea of partial pivoting is to maximise $|M_{i,j}|$ only for one of the two indices i or j and keep the other one fixed, i.e., we determine the maximal element in modulus in one particular row or one particular column.

Let i^* denote an arbitrary row index. Then the corresponding matrix row $M_{i^*,j}$ can be computed in complexity $\mathcal{O}(\#\hat{s})$. The row maximiser

$$j^* := \operatorname*{argmax}_{j\in\hat{s}} |M_{i^*,j}|, \qquad \delta := M_{i^*,j^*},$$

Algorithm 3 Cross approximation with partial pivoting and complexity $\mathcal{O}(k^2(\#\hat{t} + \#\hat{s}))$.

The input of the algorithm is the matrix $M \in \mathbb{R}^{\hat{t}\times\hat{s}}$. The output is a rank k approximation $\sum_{\nu=1}^{k} a^\nu (b^\nu)^T$ of M.

Set $i^* := \min\{i \in \hat{t}\}$, $\nu := 1$ and $\mathcal{P} := \{\}$ (the set of row pivot indices)
while $\nu \leq k$ **do**
Compute the maximal entry in modulus

$$j^* := \operatorname{argmax}_{j\in\hat{s}} |M_{i^*,j}|, \qquad \delta := M_{i^*,j^*} - \sum_{\mu=1}^{\nu-1} (a^\mu)_{i^*} (b^\mu)_{j^*}.$$

if $\delta = 0$ **then**
if $\#\mathcal{P} = n - 1$ **then**
the algorithm terminates with the exact rank $\nu - 1$ representation $R_{\nu-1}$ of the input matrix M.
end if
else
Compute the entries of the vectors a^ν, b^ν:

$$\begin{aligned} (a^\nu)_i &:= M_{i,j^*} - \sum_{\mu=1}^{\nu-1} (a^\mu)_i (b^\mu)_{j^*}, \qquad i \in \hat{t}, \\ (b^\nu)_j &:= \left(M_{i^*,j} - \sum_{\mu=1}^{\nu-1} (a^\mu)_{i^*} (b^\mu)_j \right) / \delta, \qquad j \in \hat{s}. \end{aligned}$$

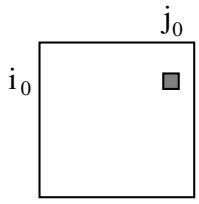
end if
$\mathcal{P} := \mathcal{P} \cup \{i^*\}$
Choose $i^* \in \hat{t} \setminus \mathcal{P}$, e.g., $i^* := \operatorname{argmax}_{i\in\hat{t}\setminus\mathcal{P}} |(b^\nu)_{i,j^*}|$
end while

yields the pivot pair (i^*, j^*) which is not a maximiser for all pairs of indices but at least for one row. The whole cross approximation procedure based on partial pivoting is given in Algorithm 3.

For $\delta \neq 0$ the arguments of Lemma 4.4 apply and the rank of the remainder is reduced by one. However, the condition $\delta \neq 0$ is sometimes a bottleneck, as the following example shows.

Example 4.7 *Let $M \in \mathbb{R}^{n\times n}$ with*

$$M_{i,j} := \begin{cases} 1 & (i,j) = (i_0, j_0) \\ 0 & \text{otherwise.} \end{cases}$$



Obviously $M \in \mathcal{R}(1, n, n)$, but if the position (i_0, j_0) of the only non-zero entry is not known in advance, then it takes in average $\mathcal{O}(n)$ steps to find the row index i_0 and in each step we have to compute n entries of a row. The average total complexity is $\mathcal{O}(n^2)$.

We conclude that the cross approximation with partial pivoting is not a useful technique for sparse matrices. For the typically dense boundary element matrices however, the method can be well-suited.

4.5 Adaptive Cross Approximation (ACA)

A variant of the cross approximation algorithms (Algorithm 2 and 3) is derived if we determine the rank k adaptively for a given approximation accuracy ε. The important question is: how do we estimate the (relative or absolute) approximation error?

A good heuristic is to estimate the remainder $\|M - R_k\|$ by a rank one approximation R. Since we use a successive rank one approximation (in each step we compute a rank one approximation of the remainder) we can estimate

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

The two vectors a^k and b^k are constructed in Algorithms 2 and 3, so the functionals

$$\begin{aligned} \epsilon_{\mathrm{abs}}(k) &:= \|a^k\|_2 \|b^k\|_2, \\ \epsilon_{\mathrm{rel}}(k) &:= \|a^k\|_2 \|b^k\|_2 / \|a^1\|_2 \|b^1\|_2 \end{aligned}$$

can be used to estimate the (absolute or relative) error $\|M - R_k\|_2$ in the Euclidean norm. For the Frobenius norm one uses

$$\begin{aligned} \epsilon^F_{\mathrm{abs}}(k) &:= \|a^k\|_2 \|b^k\|_2, \\ \epsilon^F_{\mathrm{rel}}(k) &:= \|a^k\|_2 \|b^k\|_2 / \sqrt{\sum_{\nu=1}^{k} \|a^\nu\|_2^2 \|b^\nu\|_2^2}. \end{aligned}$$

The stopping criterion "$\nu \leq k$" is replaced by

$$\epsilon_{\mathrm{rel}}(\nu - 1) \leq \varepsilon \quad \text{or} \quad \epsilon_{\mathrm{abs}}(\nu - 1) \leq \varepsilon \tag{4.1}$$

for one of the above error estimators (Euclidean or Frobenius). The initial value is $\epsilon(0) := \infty$. Since the rank is determined adaptively, this variant is called *adaptive cross approximation* (ACA). The counterexamples that will follow show that for smooth kernels g it can nonetheless happen that $\epsilon(\nu) \to 0$ although the relative or absolute approximation error is $\mathcal{O}(1)$.

In the literature [2, 1, 4] there exist some "proofs" for the convergence of (adaptive) cross approximation in the case that the matrix entries $M_{i,j}$ stem from the evaluation of a smooth function $f(x_i, y_j) = M_{i,j}$ at some points $x_i, y_j \in \mathbb{R}^d$. At the core the "proofs" are based on a theorem that states that a Lagrange interpolation with p points (in some bounded subset $D_X \subset \mathbb{R}^d$) has a uniformly good approximation quality $\eta^{\sqrt[d]{p}}, \eta < 1$, for a smooth function $f : \mathbb{R}^d \to \mathbb{R}$. Obviously, one needs assumptions on the distribution of the p points used for the interpolation, i.e., we need (geometric) assumptions on the choice of the pivot elements i^*, j^*. In particular, for all boundary element discretisations the set D_X lies on the boundary Γ (measure zero) of a domain $\Omega \subset \mathbb{R}^d$ so that the theorem can not be applied. This is not just a technical detail in the proof, as the following examples reveal.

Example 4.8 (Counterexample for cross approximation) *We define the function*

$$g : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}, \qquad (x, y) \mapsto (y_1 - x_1 - 1) \cdot (y_2 - x_2 - 1) \cdot \log \|x - y\|.$$

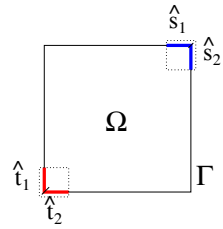
The derivatives of g are bounded by those of $\log \|x - y\|$, therefore g is asymptotically smooth, in particular in the domain $\Omega := [0, 1]^2 \times [0, 1]^2$ and on the boundary $\Gamma := \partial\Omega$.

We apply a Nystrøm discretisation

$$M_{i,j} := g(\xi_i, \xi_j)$$

for a uniformly distributed set of points $\Xi := \{\xi_1, \ldots, \xi_N\}$, $\mathcal{I} := \{1, \ldots, N\}$, on the boundary Γ (just for the sake of simplicity, in general one should apply a Galerkin discretisation). For a given admissibility parameter η we define $\delta := \eta/(1 + \eta)$ and

$$\begin{aligned}\hat{t}_1 &:= \{i \in \mathcal{I} \mid (\xi_i)_1 \in [0,\delta]\},\\ \hat{t}_2 &:= \{i \in \mathcal{I} \mid (\xi_i)_2 \in [0,\delta]\},\\ \hat{s}_1 &:= \{i \in \mathcal{I} \mid (\xi_i)_2 \in [1-\delta,1]\},\\ \hat{s}_1 &:= \{i \in \mathcal{I} \mid (\xi_i)_1 \in [1-\delta,1]\}.\end{aligned}$$

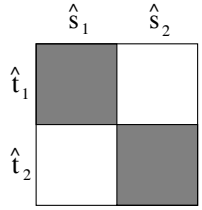


The two clusters $\hat{t} := \hat{t}_1 \cup \hat{t}_2$ *and* $\hat{s} := \hat{s}_1 \cup \hat{s}_2$ *are* η*-admissible:*

$$\operatorname{diam}(\hat{t}) = \sqrt{2}\delta = \sqrt{2}\eta/(1+\eta), \qquad \operatorname{dist}(\hat{t},\hat{s}) = \sqrt{2}(1-\delta) = \sqrt{2}/(1+\eta).$$

For each $i \in \hat{t}_1$ *and* $j \in \hat{s}_2$ *the matrix entry is*

$$M_{i,j} = g(\xi_i,\xi_j) = (1-0-1)\cdot((\xi_j)_2 - (\xi_i)_2) - 1)\cdot \log\|\xi_i - \xi_j\| = 0.$$



The same holds for $i \in \hat{t}_2$ *and* $j \in \hat{s}_1$*. The matrix block* $M|_{\hat{t}\times\hat{s}}$ *is therefore of the* 2×2 *block structure depicted rightward.*

In each of the two subblocks $\hat{t}_1 \times \hat{s}_1$ *and* $\hat{t}_2 \times \hat{s}_2$ *the matrix can be approximated by low rank (e.g. by interpolation of* g *in* $[0,\delta]^2 \times [1-\delta,1]^2$ *as in the previous section).*

The cross approximation algorithm with partial pivoting starting with an index $i^* \in \hat{t}_1$ *will compute a nonzero pivot element* $j^* \in \hat{s}_1$*. Hence, the first cross* $a^1(b^1)^T$ *is zero outside of* $\hat{t}_1 \times \hat{s}_1$ *so that the remainder is of the same structure as* M*. If we apply the standard heuristic* $i^* := \operatorname{argmax}_{i\in\hat{t}\setminus\mathcal{P}} |(b^\nu)_{i,j^*}|$ *for the next pivot row index, then again* $i^* \in \hat{t}_1$*. All pivot indices* (i^*,j^*) *will belong to* $\hat{t}_1 \times \hat{s}_1$ *until* $k > \min\{\#\hat{t}_1, \#\hat{s}_1\}$*. The second block* $M|_{\hat{t}_2\times\hat{s}_2}$ *will not be approximated at all:* $\|M - \sum_{\nu=1}^k a^\nu (b^\nu)^T\|_2 \geq \|M|_{\hat{t}_2\times\hat{s}_2}\|_2$*.*

The problems in the previous example can be remedied by taking a different heuristic for the choice of pivot elements leading to the ACA+ algorithm that will be introduced later.

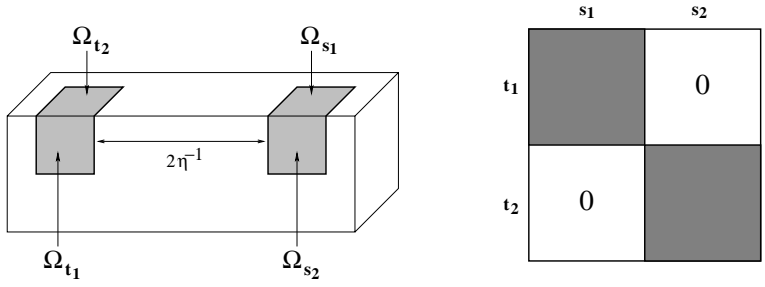
The counterexample seems to be artificial, since the kernel function was chosen so that (partially pivoted) cross approximation fails. In the next example the same effect appears for the standard double layer potential in three dimensions. There the kernel function is only asymptotically smooth with respect to one of the two variables.

Example 4.9 (DLP counterexample for cross approximation) *We consider the Nyström discretisation of the double layer potential* $g(x,y) = \frac{\langle x-y, n(y)\rangle}{4\pi\|x-y\|^3}$ *in three dimensions for an* η*-admissible block* $t \times s$*. The corresponding domains of the two clusters are*

$$\Omega_t = \Omega_{t_1} \cup \Omega_{t_2}, \qquad \Omega_s = \Omega_{s_1} \cup \Omega_{s_2},$$

where

$$\begin{aligned}\Omega_{t_1} &:= [0,1]\times[0,1]\times\{0\},\\ \Omega_{t_2} &:= [0,1]\times\{0\}\times[0,1],\\ \Omega_{s_1} &:= [1+2\eta^{-1}, 2+2\eta^{-1}]\times\{0\}\times[0,1],\\ \Omega_{s_2} &:= [1+2\eta^{-1}, 2+2\eta^{-1}]\times[0,1]\times\{0\}.\end{aligned}$$



The matrix block M has entries

$$M_{i,j} := g(x_i, y_j), \qquad x_i \in \Omega_t, y_j \in \Omega_s.$$

The normal vector $n(y)$ is perpendicular to the xy-plane, and for all $x_i \in \Omega_{t_1}, y_j \in \Omega_{s_2}$ the difference $x_i - y_j$ lies in the xy-plane. Therefore,

$$M_{i,j} = 0 \qquad \text{for all } i \in \hat{t}_1, j \in \hat{s}_2 \quad \text{and} \quad i \in \hat{t}_2, j \in \hat{s}_1.$$

The double layer kernel $g(x, y)$ is a standard kernel function that is asymptotically smooth with respect to the first variable x but not with respect to the second variable y (there are jumps in $n(y)$ across edges).

Let us assume that in Algorithm 3 the first pivot index i^* belongs to $\hat{t}_1$. Then the corresponding row has zero entries for all $j \in \hat{s}_2$, so that the column pivot index j^* (row maximiser) belongs to $\hat{s}_1$. The cross

$$R_1 = a^1(b^1), \quad a^1_i = M_{i,j^*}, \quad b^1_j = M_{i^*,j}/M_{i^*,j^*}$$

has non-zero entries only for all $(i, j) \in \hat{t}_1 \times \hat{s}_1$. This means that the off-diagonal zero blocks are unmodified during the update, i.e., $M - R_1$ is of the same block structure as M. Consequently, if all pivot rows are chosen from $\hat{t}_1$ (which is the case for the default choice $i^* := \operatorname{argmax}_i |(b^\nu)_{i,j^*}|$ in Algorithm 3), then

$$\|(M - R_k)|_{\hat{t}_1 \times \hat{s}_1}\|_2 \to 0 \quad \text{but} \quad \|(M - R_k)|_{\hat{t}_2 \times \hat{s}_2}\|_2 \sim \|M|_{\hat{t}_2 \times \hat{s}_2}\|_2 \neq 0 \quad \text{for all } k \leq \#\hat{t}_1.$$

The partial pivot search simply fails to "see" the second block $M|_{\hat{t}_2 \times \hat{s}_2}$. There occurs no breakdown during the iteration in Algorithm 3 but still the approximation is bad.

Conclusion: The standard cross approximation algorithms from the literature fail even for smooth kernels and the standard double layer kernel on domains with edges.

4.6 Improved Adaptive Cross Approximation (ACA+)

As the many counterexamples in the previous sections indicate, it is not advisable to use ACA for the assembly of stiffness matrices from boundary integral operators. In the next section we present a hybrid method (HCA) for the assembly of admissible matrix blocks. That method however is not purely algebraic, that means it is not sufficient to provide only a function $(i, j) \mapsto M_{i,j}$, we also need to evaluate the kernel function $(x, y) \mapsto g(x, y)$.

In practice we are sometimes forced to assemble the BEM stiffness matrix in a grey-box fashion. That means we are given the locations of the basis functions, e.g., a nodal point ξ_i from the support of φ_i (we need this to construct the cluster tree $T_{\mathcal{I}}$ and block cluster tree $T_{\mathcal{I}\times\mathcal{I}}$), but the assembly of matrix entries $M_{i,j}$ is done by some provided procedure `assemble(i, j)`. In this setting we can use either the expensive SVD (not reasonable for large scale problems) or a heuristic like ACA. Due to the fact that ACA fails even for the standard double layer potential operator on domains with edges, we will improve the pivoting strategy of ACA in this section. This variant of ACA is called the ACA+ algorithm.

We denote the matrix to be approximated by $M \in \mathbb{R}^{n\times m}$. A reference column of M is defined by

$$(a^{\text{ref}})_i \quad := \quad M_{i,j_{\text{ref}}}, \qquad i \in \{1, \ldots, n\},$$

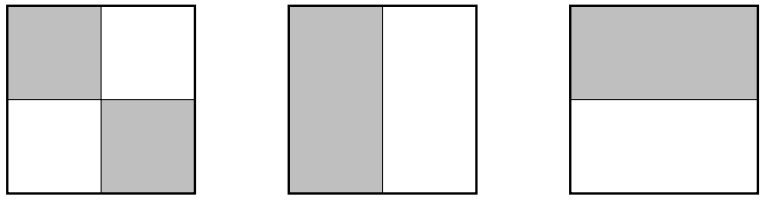


Figure 4.4: Three situations where ACA (with standard partial pivoting) fails.

where the choice of the index j_{ref} is arbitrary. The reference row is determined by

$$\begin{aligned} i_{\text{ref}} &:= \text{argmin}_{i\in\{1,\dots,n\}}|(a^{\text{ref}})_i|, \\ (b^{\text{ref}})_j &:= M_{i_{\text{ref}},j}, \qquad j \in \{1,\dots,m\}. \end{aligned}$$

The choice of i_{ref} is counterintuitive, because i_{ref} corresponds to a row that produces a small entry in the column j_{ref}. The reason why we do this is that we expect one of the three situations from Figure 4.4 where ACA with standard partial pivoting will fail. The above choice of i_{ref} and j_{ref} ensures each non-zero block is intersected by a reference row or column. The role of the reference row and column is that of an observer that tells us where to start with the pivot search.

The pivoting strategy is now based on the knowledge that we gain from a^{ref} and b^{ref}:

1. Determine the index i^* of the largest entry in modulus in a^{ref}, $i^* := \text{argmax}_{i=1,\dots,n}\,|a_i^{\text{ref}}|$.
2. Determine the index j^* of the largest entry in modulus in b^{ref}, $j^* := \text{argmax}_{j=1,\dots,m}\,|b_j^{\text{ref}}|$.
3. If $|a_{i^*}^{\text{ref}}| > |b_{j^*}^{\text{ref}}|$ then
   (a) compute the vector $b \in \mathbb{R}^m$ with entries $b_j := M_{i^*,j}$;
   (b) redefine the column pivot index $j^* := \text{argmax}_{j=1,\dots,m}\,|b_j|$;
   (c) compute the vector $a \in \mathbb{R}^n$ with entries $a_i := M_{i,j^*}/M_{i^*,j^*}$;
4. otherwise
   (a) compute the vector $a \in \mathbb{R}^n$ with entries $a_i := M_{i,j^*}$;
   (b) redefine the row pivot index $i^* := \text{argmax}_{i=1,\dots,n}\,|a_i|$;
   (c) compute the vector $b \in \mathbb{R}^m$ with entries $b_j := M_{i^*,j}/M_{i^*,j^*}$.
5. The rank one matrix ab^T is the sought cross approximation to M.

In the next step we can omit to determine a new reference row and colum by updating the already existing ones. Of course, it is not reasonable to use the reference row i_{ref} if it was chosen as a pivot index, i.e., $i^* = i_{\text{ref}}$ or to use the reference column if $j^* = j_{\text{ref}}$ (the corresponding matrix row (column) of the remainder is zero). In that case we have to define a new reference row or column (provided the stopping criterion by a given rank k or given accuracy ε is not fulfilled). We denote the already used pivot elements by $\mathcal{P}_{\text{rows}}$ and $\mathcal{P}_{\text{cols}}$ and proceed as follows:

1. We update the reference row and column

$$a^{\text{ref}} := a^{\text{ref}} - a \cdot b_{j_{\text{ref}}}, \qquad b^{\text{ref}} := b^{\text{ref}} - a_{i_{\text{ref}}} \cdot b.$$

2. If both $i^* = i_{\text{ref}}$ and $j^* = j_{\text{ref}}$, then we have to determine both of them as above, where we restrict the possible indices to non-pivot ones.

3. If $i^* = i_{\text{ref}}$ and $j^* \neq j_{\text{ref}}$, then we have to choose a new reference row b^{ref} corresponding to a reference index i_{ref} that has not yet been a pivot index and that is consistent to j_{ref}:

$$i_{\text{ref}} := \operatorname*{argmin}_{i \in \{1,\dots,n\} \setminus \mathcal{P}_{\text{rows}}} |a_i^{\text{ref}}|$$

4. If $j^* = j_{\text{ref}}$ and $i^* \neq i_{\text{ref}}$, then we have to choose a new reference column a^{ref} corresponding to a reference index j_{ref} that has not yet been a pivot index and that is consistent to i_{ref}:

$$j_{\text{ref}} := \operatorname*{argmin}_{j \in \{1,\dots,m\} \setminus \mathcal{P}_{\text{cols}}} |b_j^{\text{ref}}|$$

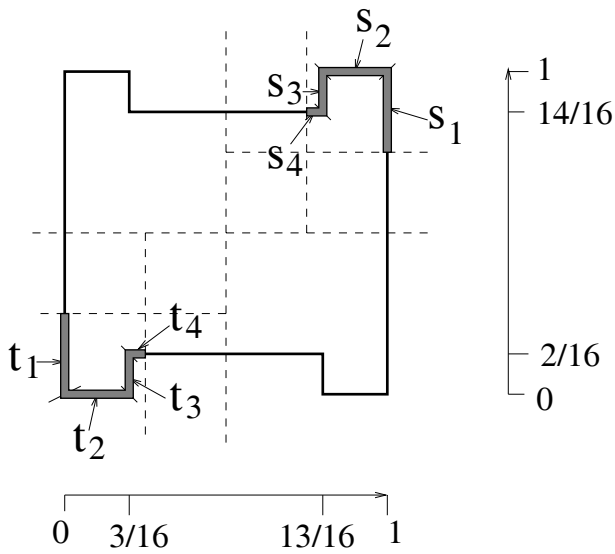
For the improved cross approximation ACA+ we are not able to prove convergence, but at least for the counterexamples presented above the method works fine, because we have modified the partial pivot search correspondingly. The question is: Is it possible to find a partial pivoting strategy that yields a good cross approximation in $\mathcal{O}(n)$? The following example gives a negative answer to that question.

Example 4.10 (Counterexample for partial pivoting) *We define the functions*

$$\begin{aligned} p(z) &:= (z-1)\cdot(z-13/16), \\ q(z) &:= z\cdot(z-1/8), \\ r(z) &:= (z-1)\cdot(z-7/8) \end{aligned}$$

and

$$f : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}, \quad (x,y) \mapsto p(x_1 - y_1)\cdot q(x_2)\cdot r(y_2).$$



The polynomials p, q, r are chosen such that

$$q(x_2) \equiv 0 \quad \text{for } x \in t_2 \cup t_4, \tag{4.2}$$

$$r(y_2) \equiv 0 \quad \text{for } y \in s_2 \cup s_4, \tag{4.3}$$

$$p(x_1 - y_1) \equiv 0 \quad \text{for } (x,y) \in t_1 \times s_1 \cup t_3 \times s_1 \cup t_1 \times s_3. \tag{4.4}$$

We define the matrix $M \in \mathbb{R}^{n\times m}$ by

$$M_{i,j} := f(x_i, y_j),$$

where the points $x_i \in t = t_1 \cup t_2 \cup t_3 \cup t_4$ and $y_j \in s = s_1 \cup s_2 \cup s_3 \cup s_4$ are uniformly distributed in $t, s \subset \Gamma = \partial\Omega$ of the H-domain Ω depicted above. From (4.2) and (4.3) and (4.4) it follows that M is of the

block structure

$$M = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & M^{3,3} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

where $M^{3,3} \in \mathbb{R}^{\#t_3 \times \#s_3}$. *Here, cross approximation with partial pivoting faces the same problem as in Example 4.7: the block* $M^{3,3}$ *can only be found in quadratic complexity* $\mathcal{O}(nm)$. *On the other hand, polynomial interpolation yields the exact representation with separation rank* $k = (m+1)^2 = 4$. *Of course, we can add a "nice" smooth function to* f *in order to avoid the zero matrix entries and the breakdown of the cross approximation.*

Conclusion: The grey-box algorithm ACA+ seems to work for many practical cases, in particular for the double layer potential operator on domains with edges. In general however, it is impossible to find reliable partial pivoting strategies and it is even impossible for the algorithm to detect whether it has succeeded or not — unless one checks all $n \times m$ matrix entries.

4.7 Hybrid Cross Approximation (HCA)

The problems for ACA (and ACA+) stem from the fact that we evaluate the kernel function only in a part of the boundary Γ (measure zero) and not in the whole volume. Additionally, the results of the existing convergence proofs are not convincing and we don't have a reliable error estimate (there are claims and "proofs" but in presence of the above examples they are at best misleading).

The idea for *hybrid cross approximation* is to use the nodal points of a standard tensor-product Chebyshev interpolation scheme in a bounding box Q_t that contains a given cluster t, as pivot points for cross approximation. The m-th order tensor interpolation in $Q_t \subset \mathbb{R}^d$ uses $(m+1)^d$ interpolation points $(x_\nu^t)_{\nu \in K}$. We define the coupling matrix $S \in \mathbb{R}^{K \times K}$ by

$$S_{\nu,\mu} := g(x_\nu^t, x_\mu^s)$$

and observe that for an admissible block cluster $t \times s$ the interpolation based low-rank approximation is of the form

$$\tilde{M}|_{\hat{t} \times \hat{s}} = U^t S (V^s)^T,$$

where the entries of U^t, V^s are

$$U^t_{i,\nu} := \int_\Omega \varphi_i(x) L^t_\nu(x) \mathrm{d}x, \qquad V^s_{j,\mu} := \int_\Omega \varphi_j(x) L^s_\mu(y) \mathrm{d}y.$$

The matrix $S \in \mathbb{R}^{K \times K}$ is the pointwise evaluation of an asymptotically smooth kernel g at nodal points distributed in the whole volume of the bounding box product $Q_t \times Q_s$, therefore we expect ACA (with partial or full pivoting) to work for this type of matrix. In particular, S is of size $K \times K$ so that full pivoting and a posteriori error control is feasible. Let $\mathcal{P}_{\mathrm{rows}}$ denote the row pivot indices and $\mathcal{P}_{\mathrm{cols}}$ the column pivot indices. A cross approximation of S is of the form (Lemma 4.6)

$$S \approx S_k := S|_{K \times \mathcal{P}_{\mathrm{cols}}} \cdot G \cdot S|_{\mathcal{P}_{\mathrm{rows}} \times K}, \qquad G = (S|_{\mathcal{P}_{\mathrm{rows}} \times \mathcal{P}_{\mathrm{cols}}})^{-1} \in \mathbb{R}^{\mathcal{P}_{\mathrm{cols}} \times \mathcal{P}_{\mathrm{rows}}}.$$

The entries of the product $U^t S|_{K \times \mathcal{P}_{\mathrm{cols}}}$ are

$$(U^t S|_{K \times \mathcal{P}_{\mathrm{cols}}})_{i\mu} = \sum_{\nu \in K} U^t_{i,\nu} g(x_\nu^t, x_\mu^s) = \sum_{\nu \in K} \int_\Omega \varphi_i(x) L_\nu(x) \mathrm{d}x g(x_\nu^t, x_\mu^s) = \int_\Omega \varphi_i(x) \mathcal{I}^t_m g(x, x_\mu^s) \mathrm{d}x.$$

If we now replace the interpolation $\mathcal{I}^t_m g(x, x_\mu^s)$ by the exact kernel g, we get

$$\tilde{M}|_{\hat{t} \times \hat{s}} \approx R_k := AGB^T, \qquad A \in \mathbb{R}^{\hat{t} \times \mathcal{P}_{\mathrm{cols}}}, \qquad B \in \mathbb{R}^{\hat{s} \times \mathcal{P}_{\mathrm{rows}}},$$

where the two matrices A and B consist of collocation integrals

$$A_{i,\nu} := \int_\Omega \varphi_i(x) g(x, x_\nu^s) \mathrm{d}x, \qquad B_{j,\mu} := \int_\Omega \varphi_j(y) g(x_\mu^t, y) \mathrm{d}y$$

and the matrix G is

$$G = \left(S|_{\mathcal{P}_{\text{rows}} \times \mathcal{P}_{\text{cols}}}\right)^{-1}.$$

The entries of $R_k = AGB^T$ coincide with

$$(\widehat{R}_k)_{i,j} := \int_\Omega \int_\Omega \varphi_i(x) \tilde{g}(x,y) \varphi_j(y) \mathrm{d}x \mathrm{d}y,$$

where the degenerate kernel $\tilde{g}$ is given by

$$\tilde{g}(x,y) := \sum_{\nu \in \mathcal{P}_{\text{rows}}} \sum_{\mu \in \mathcal{P}_{\text{cols}}} g(x, x_\nu^s) G_{\nu,\mu} g(x_\mu^t, y). \tag{4.5}$$

From the representation of R_k we can see that the interpolation is only used to determine the correct pivot indices. However, since we restrict the set of allowed pivot indices, the accuracy $\|M - Rk\|$ will depend on both the accuracy of the interpolation $M - \tilde{M}$ as well as the accuracy of the cross approximation $\|S - S_k\|$.

Theorem 4.11 *Let $Q_t \times Q_s$ the product of the two bounding boxes for two clusters $t, s \in T_{\mathcal{I}}$. The degenerate kernel $\tilde{g}$ from (4.5) fulfils*

$$\forall (x,y) \in Q_t \times Q_s : \quad |g(x,y) - \tilde{g}(x,y)| \leq \epsilon_{int}(g) + \epsilon_{int}(\tilde{g}) + \Lambda_m^{2d} \epsilon_{ACA},$$

where $\epsilon_{int}(g)$ and $\epsilon_{int}(\tilde{g})$ are the interpolation errors of g and $\tilde{g}$ by the m-th order tensor Lagrange interpolation

$$\begin{aligned} g^{\text{int}}(x,y) &:= \sum_{\nu \in K} \sum_{\mu \in K} L_\nu^t(x) g(x_\nu^t, y_\mu^s) L_\mu^s(y) \\ \tilde{g}^{\text{int}}(x,y) &:= \sum_{\nu \in K} \sum_{\mu \in K} L_\nu^t(x) \tilde{g}(x_\nu^t, y_\mu^s) L_\mu^s(y), \end{aligned}$$

with stability constant Λ_m, and ϵ_{ACA} is the entrywise error of the cross approximation of the coupling matrix S: $|(S_k - S)_{ij}| \leq \epsilon_{ACA}$.

Proof: [13, Corollary25] ■

If the kernel function g is of the form

$$g(x,y) = D_x D_y \gamma(x,y)$$

for some partial differential operators D_x acting on x and D_y acting on y, then Theorem 4.11 cannot be applied directly: g might not be smooth in any of the two variables. We assume that the so-called *generating kernel function* γ is asymptotically smooth. Then we define the approximating separable kernel $\tilde{g}$ by

$$\tilde{g}(x,y) := D_x D_y \tilde{\gamma} = \sum_{\nu \in \mathcal{P}_{\text{rows}}} \sum_{\mu \in \mathcal{P}_{\text{cols}}} D_x g(x, x_\nu^s) G_{\nu,\mu} D_y g(x_\mu^t, y) \tag{4.6}$$

so that the low rank approximation of $M|_{\hat{t} \times \hat{s}}$ by $\tilde{M}|_{\hat{t} \times \hat{s}} = AGB^T$ consists of G as above and matrices A, B with entries

$$A_{i,\nu} := \int_\Omega \varphi_i(x) D_x g(x, x_\nu^s) \mathrm{d}x, \qquad B_{j,\mu} := \int_\Omega \varphi_j(y) D_y g(x_\mu^t, y) \mathrm{d}y.$$

The approximation error can be estimated analogously to Theorem 4.11 where the error $|D_x D_y \gamma - D_x D_y \tilde{\gamma}|$ is now amplified by the derivatives [13, Theorem26].

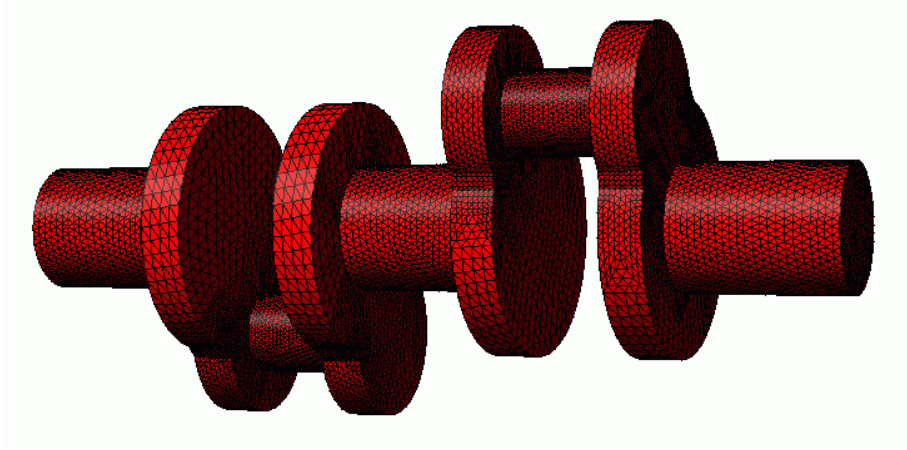


Figure 4.5: The crank shaft geometry

Remark 4.12 *The error of the cross approximation is amplified by Λ_m^{2d}, but $\Lambda_m \sim \log m$ depends only logarithmically on m, i.e, on $\log(1/\varepsilon)$. The interpolation error of g has already been estimated in the previous section. In order to estimate the interpolation error for $\tilde{g}$, we have to give bounds for the asymptotic smoothness of $\tilde{g}$. From (4.5) we observe that both g and $\tilde{g}$ have the same asymptotic behaviour with respect to partial derivatives ∂_x^i and ∂_y^j. However, the coefficients $G_{\nu,\mu}$ can only be bounded by ϵ_{ACA}^{-1} so that the order of the interpolation has to be $m \sim \log(1/(\varepsilon \cdot \epsilon_{ACA})$ in order to compensate this. This is a purely theoretical artifact: we expect that the substitution of $\sum_{\nu \in K} L_\nu^t g(x_\nu^t, y)$ by $g(x,y)$ will only decrease and not increase the error, so that the expected error is*

$$\forall (x,y) \in Q_t \times Q_s : \quad |g(x,y) - \tilde{g}(x,y)| \lesssim \max\{\epsilon_{int}(g), \epsilon_{ACA}\}.$$

The numerical tests in the next section will even suggest that

$$|g(x,y) - \tilde{g}(x,y)| \leq c_1 \epsilon_{int}(g) + c_2 \epsilon_{ACA},$$

where $c_1 \ll 1$ and $c_2 \approx 1$ for the single layer potential and $c_2 \approx 30$ for the double layer potential, i.e., a weak dependency on the interpolation order and a strong dependency on the cross approximation error.

4.8 Numerical Examples in 3D for Interpolation, ACA, ACA+ and HCA

4.8.1 The Test Problem: 3D Laplace Single- and Double-Layer Potential

For the numerical examples in this section we consider as a reference problem the single layer potential operator $\mathcal{V}$ and double layer potential operator $\mathcal{K}$ of the three-dimensional Laplacian,

$$\begin{aligned} \mathcal{V}[u](x) &:= \int_\Gamma \frac{1}{4\pi \|x-y\|} u(y) dy, \quad x \in \Gamma, \\ \mathcal{K}[u](x) &:= \int_\Gamma \frac{\langle x-y, n(y) \rangle}{4\pi \|x-y\|^3} u(y) dy, \quad x \in \Gamma, \end{aligned}$$

where $\Gamma := \partial\Omega$ is the surface of the domain Ω describing a crank shaft geometry, cf. Figure 4.5 (an example from the NETGEN library of Joachim Schöberl). The surface Γ is discretised by n flat triangles of a triangulation $\mathcal{T}$. The Galerkin discretisation V_n of $\mathcal{V}$ and K_n of $\mathcal{K}$ in the piecewise constant basis $\{\varphi_i = \chi_{\tau_i} \mid \tau_i \in \mathcal{T}\}$ yields fully populated matrices that can be approximated by data-sparse $\mathcal{H}$-matrices

$\tilde{V}_n, \tilde{K}_n$. Each admissible block of the $\mathcal{H}$-matrix is assembled by either ACA, ACA+, HCA or interpolation. For the quadrature we choose a variable order that is lowered in the far-field where the kernel function is smoother. The underlying cluster tree $T_{\mathcal{I}}$ is produced by regular subdivision with stopping parameter $n_{min} := 20$, and the block cluster tree $T_{\mathcal{I}\times\mathcal{I}}$ is constructed in the canonic way using the standard admissibility with $\eta = 2$ based on bounding boxes.

All computations are performed on a single UltraSPARC IIIcu processor with 900 MHz clock rate in a SunFire6800 machine. The elapsed time is measured in seconds and the storage requirements are measured in megabyte.

4.8.2 Interpolation

First, we consider the interpolation approach. The accuracy of the approximation will depend on the order $p = m + 1$ of the interpolation. According to (3.41) we expect an exponential decay, and this is underlined by the results in Table 4.1. The complexity for the assembly and the storage requirements of the matrix entries is expected to be linear in the rank $k = p^3$ and thus cubic in p, which is in accordance with the results in Table 4.1.

	DLP $M = K_n$				SLP $M = V_n$			
p	Time	Storage	$\frac{\|M-\tilde{M}\|_2}{\|M\|_2}$	$\|I - \tilde{M}^{-1}M\|_2$	Time	Storage	$\frac{\|M-\tilde{M}\|_2}{\|M\|_2}$	$\|I - \tilde{M}^{-1}M\|_2$
1	95	255	1.4×10^{-1}	2.2×10^{-0}	93	255	9.3×10^{-2}	9.6×10^{-0}
2	150	652	3.5×10^{-2}	1.2×10^{-0}	144	652	8.2×10^{-3}	4.6×10^{-1}
3	365	1731	5.7×10^{-3}	6.3×10^{-1}	343	1731	7.0×10^{-4}	4.7×10^{-2}
4	1060	3830	8.2×10^{-4}	1.5×10^{-1}	1028	3830	6.9×10^{-5}	6.9×10^{-3}
5	1996	7291	1.9×10^{-4}	1.9×10^{-2}	1890	7291	1.5×10^{-5}	1.2×10^{-3}
6	3625	12455	3.9×10^{-5}	3.6×10^{-3}	3528	12455	1.8×10^{-6}	2.1×10^{-4}

Table 4.1: Assembly of K_n and $V_n, n = 25744$, using a p-th order interpolation scheme.

4.8.3 ACA and ACA+

The second numerical test illustrates the behaviour of ACA and ACA+ when these methods are used to assemble $\tilde{K}_n$ (cf. Example 4.9). In Table 4.2 we observe that ACA fails to assemble the stiffness matrix whereas ACA+ yields a good approximation. This behaviour is not stable. For finer discretisations of the

	ACA				ACA+			
ϵ_{ACA}	Time	Storage	$\frac{\|M-\tilde{M}\|_2}{\|M\|_2}$	$\|I - \tilde{M}^{-1}M\|_2$	Time	Storage	$\frac{\|M-\tilde{M}\|_2}{\|M\|_2}$	$\|I - \tilde{M}^{-1}M\|_2$
1×10^{-1}	131	303	1.4×10^{-2}	1.1×10^{-0}	171	351	2.9×10^{-3}	5.0×10^{-1}
1×10^{-2}	185	434	8.3×10^{-3}	3.8×10^{-1}	206	469	3.2×10^{-4}	4.8×10^{-2}
1×10^{-3}	251	584	6.2×10^{-3}	2.7×10^{-1}	249	609	4.2×10^{-5}	4.1×10^{-3}
1×10^{-4}	329	752	4.4×10^{-3}	2.1×10^{-1}	301	772	3.2×10^{-6}	4.6×10^{-4}
1×10^{-5}	422	936	4.1×10^{-3}	1.9×10^{-1}	357	958	4.4×10^{-7}	3.9×10^{-5}

Table 4.2: Assembly of $K_n, n = 25744$, using ACA and ACA+ with different stopping tolerances ϵ_{ACA}.

same geometry with $n_2 = 102976$ panels we observe that both ACA and ACA+ are deteriorated, cf. Table 4.3. The prescribed stopping tolerance ϵ_{ACA} is not met: the error estimate (4.1) becomes very small when the exact error is still rather large. For sufficiently small ϵ_{ACA} the improved adaptive cross approximation

ACA+ is able to find the desired approximation, but then both the time for the assembly and the storage requirements are undesirably large.

	ACA				ACA+			
ϵ_{ACA}	Time	Storage	$\frac{\|M-\tilde{M}\|_2}{\|M\|_2}$	$\|I-\tilde{M}^{-1}M\|_2$	Time	Storage	$\frac{\|M-\tilde{M}\|_2}{\|M\|_2}$	$\|I-\tilde{M}^{-1}M\|_2$
1×10^{-4}	5788	1900	8.9×10^{-3}	3.7×10^{-1}	5262	1874	2.1×10^{-4}	3.4×10^{-2}
1×10^{-5}	9035	6603	7.1×10^{-3}	3.7×10^{-1}	8096	6625	2.0×10^{-4}	3.4×10^{-2}
1×10^{-6}	15100	12482	8.5×10^{-3}	3.7×10^{-1}	13804	12353	2.1×10^{-4}	3.4×10^{-2}

Table 4.3: Assembly of $K_n, n = 102976$, using ACA and ACA+ with different stopping tolerances ϵ_{ACA}.

4.8.4 HCA

The third numerical test supports the statements in Remark 4.12: we fix the order $p := 3$ of the polynomial interpolation and vary the accuracy ϵ_{ACA} in the HCA algorithm. In Table 4.4 one can see that the error of the approximation by HCA tends to be much smaller than the interpolation error with order $p = 3$. For very small ϵ_{ACA} the HCA error stabilises at a certain level. For the Laplace kernel it seems that the choice $\epsilon_{ACA} = 10^{-p-1}$ is appropriate. With this choice of parameters we start the fourth numerical test where we

	DLP $M = K_n$		SLP $M = V_n$	
	$\frac{\|M-\tilde{M}\|_2}{\|M\|_2}$	$\|I-\tilde{M}^{-1}M\|_2$	$\frac{\|M-\tilde{M}\|_2}{\|M\|_2}$	$\|I-\tilde{M}^{-1}M\|_2$
Interpolation, $p = 3$	5.7×10^{-3}	6.3×10^{-1}	7.0×10^{-4}	4.7×10^{-2}
$\epsilon_{ACA} = 1\times10^{-2}$	9.4×10^{-3}	3.5×10^{-0}	2.0×10^{-3}	2.5×10^{-1}
$\epsilon_{ACA} = 1\times10^{-3}$	7.5×10^{-4}	1.5×10^{-1}	1.4×10^{-4}	2.3×10^{-2}
$\epsilon_{ACA} = 1\times10^{-4}$	3.0×10^{-4}	3.9×10^{-2}	3.2×10^{-5}	3.2×10^{-3}
$\epsilon_{ACA} = 1\times10^{-5}$	8.7×10^{-5}	1.2×10^{-2}	1.2×10^{-5}	1.5×10^{-3}
$\epsilon_{ACA} = 1\times10^{-6}$	6.5×10^{-5}	9.4×10^{-3}	1.1×10^{-5}	1.4×10^{-3}

Table 4.4: We assemble the low rank blocks of the matrix $\tilde{M} \in \{\tilde{\mathcal{K}}_n, \tilde{\mathcal{V}}_n\}$ using HCA with fixed interpolation order $p = 3$ and different accuracies ϵ_{ACA} for ACA applied to the coupling matrices S.

approximate V_n and K_n by HCA with different accuracies ϵ_{ACA}. The results in Table 4.5 underline that HCA converges faster than standard interpolation but with a complexity comparable to ACA. However, for large p (i.e., small ϵ_{ACA}) the full pivoting for the $p^3 \times p^3$ coupling matrix $S^{t,s}$ in each block $M|_{\hat{t}\times\hat{s}}$ is dominant. This can be overcome by using a partial pivoting strategy — at least for the Laplace kernel.

	DLP $M = K_n$				SLP $M = V_n$			
ϵ_{ACA}	Time	Storage	$\frac{\|M-\tilde{M}\|_2}{\|M\|_2}$	$\|I-\tilde{M}^{-1}M\|_2$	Time	Storage	$\frac{\|M-\tilde{M}\|_2}{\|M\|_2}$	$\|I-\tilde{M}^{-1}M\|_2$
1×10^{-2}	141	485	1.0×10^{-2}	3.4×10^{-0}	138	484	2.0×10^{-3}	2.6×10^{-1}
1×10^{-3}	172	678	6.8×10^{-4}	1.5×10^{-1}	166	677	1.3×10^{-4}	2.6×10^{-2}
1×10^{-4}	269	895	2.2×10^{-4}	2.7×10^{-2}	263	880	2.6×10^{-5}	2.8×10^{-3}
1×10^{-5}	486	1143	1.7×10^{-5}	2.2×10^{-3}	472	1101	2.2×10^{-6}	3.2×10^{-4}
1×10^{-6}	1096	1288	1.2×10^{-6}	1.0×10^{-4}	1075	1171	2.2×10^{-7}	3.0×10^{-5}

Table 4.5: Assembly of K_n and V_n using an HCA approximation with accuracy parameter ϵ_{ACA}.

The last numerical test in this section compares the three methods interpolation, ACA+ and HCA to each other when used on three different discretisation levels with $n_1 = 25744$, $n_2 = 102976$ and $n_3 = 411904$

panels. The order of the quadrature is adapted to the discretisation level so that the relative inversion error is $\epsilon_1 \approx 10^{-2}$, $\epsilon_2 \approx 3 \times 10^{-3}$ and $\epsilon_3 \approx 10^{-3}$. Each of the compression schemes uses parameters that yield an approximation with relative inversion error approximately ϵ_i (ACA+ fails to achieve this for the two larger problems and the interpolation approach is too expensive for the largest problem). The results are given in Table 4.6. For all three discretisation levels we have applied an on-the-fly recompression and coarsening scheme that will be introduced later. This keeps the storage requirements almost independent of the method for the assembly and at the same time minimises the total amount of storage needed.

	$n_1 = 25744$		$n_2 = 102976$		$n_2 = 411904$	
Method	Time	Storage	Time	Storage	Time	Storage
ACA+	656	235	(6538)	1499	(88811)	8003
Interpolation	3416	235	50054	1485	918000	—
HCA	587	235	3203	1484	34540	8025

Table 4.6: Assembly of $\mathcal{K}_n$ on three different levels with $n \in \{25744, 102976, 411904\}$ panels. The time in brackets is for a constant order quadrature instead of the variable-order one (roughly twice as expensive).

Chapter 5

Elliptic Partial Differential Equations

In the previous two sections we have introduced the hierarchical matrix format which is well suited for the fast assembly and evaluation of stiffness matrices stemming from non-local integral operators. Since the stiffness matrix of those operators is in general dense, one has to apply a compression method in order to avoid the $\mathcal{O}(n^2)$ complexity.

The case is different for the discretisation and solution of partial differential equations. Differential operators are local so that a finite element discretisation leads to a *sparse* stiffness matrix. This means, the assembly and evaluation is of (optimal) complexity $\mathcal{O}(n)$. However, the inverse to the stiffness matrix is in general dense. In Theorem 5.18 (cf. [3]) we prove that the inverse allows for a data-sparse $\mathcal{H}$-matrix approximation. The analysis shows that no smoothness of the coefficients need to be required.

5.1 Sparse Finite-Element Matrices

We consider the differential equation

$$\begin{aligned} L[u](x) &:= -\operatorname{div}(C \operatorname{grad} u)(x) \\ &= \sum_{i,j=1}^{d} \frac{\partial}{\partial x_i}\left(C_{ij}(x)\frac{\partial}{\partial x_j}u(x)\right) = f(x) \qquad \text{for all } x \in \Omega \subset \mathbb{R}^d \end{aligned} \tag{5.1}$$

with Dirichlet boundary conditions

$$u(x) = 0 \qquad \text{for all } x \in \Gamma := \partial\Omega,$$

where $C(x) = (C_{ij}(x))_{i,j=1}^{d}$ is a symmetric and positive definite matrix for all $x \in \Omega$ and $f \in L^2(\Omega)$ is the right-hand side.

The corresponding weak formulation reads as follows:

Seek the solution $u \in H_0^1(\Omega)$ of

$$\int_\Omega \langle \operatorname{grad} v(x), C(x) \operatorname{grad} u(x)\rangle \,\mathrm{d}x = \int_\Omega v(x) f(x)\,\mathrm{d}x \qquad \text{for all } v \in H_0^1(\Omega). \tag{5.2}$$

We define

$$\begin{aligned} a : H_0^1(\Omega) \times H_0^1(\Omega) \to \mathbb{R}, &\qquad (v,u) \mapsto \int_\Omega \langle \operatorname{grad} v(x), C(x)\operatorname{grad} u(x)\rangle \,\mathrm{d}x, \\ F : H_0^1(\Omega) \to \mathbb{R}, &\qquad v \mapsto \int_\Omega v(x) f(x)\,\mathrm{d}x, \end{aligned}$$

and can now write (5.2) in the short form

$$a(v,u) = F(v) \qquad \text{for all } v \in H_0^1(\Omega).$$

Let $V_n := \operatorname{span}\{\varphi_1, \ldots, \varphi_n\}$ be an n-dimensional subspace of $H_0^1(\Omega)$, and let $\mathcal{I} := \{1, \ldots, n\}$. The coefficient vector $\mathbf{u} = (u_i)_{i\in\mathcal{I}} \in \mathbb{R}^{\mathcal{I}}$ of the Galerkin solution $u_n(x) := \sum_{i\in\mathcal{I}} u_i\varphi_i(x)$ is the solution to the linear system

$$A\mathbf{u} = \mathbf{f} \tag{5.3}$$

for the stiffness matrix $A = (A_{ij})_{i,j\in\mathcal{I}} \in \mathbb{R}^{\mathcal{I}\times\mathcal{I}}$ and the load vector $\mathbf{f} = (f_i)_{i\in\mathcal{I}} \in \mathbb{R}^{\mathcal{I}}$ given by

$$\begin{aligned} A_{ij} &:= a(\varphi_i, \varphi_j) = \int_\Omega \langle \operatorname{grad}\varphi_i(x), C(x)\operatorname{grad}\varphi_j(x)\rangle \,\mathrm{d}x && \text{for all } i,j \in \mathcal{I}, \\ f_i &:= F(\varphi_i) = \int_\Omega \varphi_i(x) f(x)\,\mathrm{d}x && \text{for all } i \in \mathcal{I}. \end{aligned} \tag{5.4}$$

In the case of a general Galerkin method, the matrix A may be fully populated (when all φ_i have global support). The finite-element method, however, uses basis function with minimal support. Typical are the piecewise affine basis functions on a triangulation $\mathcal{T}$ of Ω, with one Lagrange basis function φ_i associated to each nodal point x_i of the triangulation. In such a case, A is sparse. In particular, $A_{ij} \neq 0$ holds only if $i = j$ or if the nodal points x_i and x_j are connected by an edge of a triangle $\tau \in \mathcal{T}$.

As in Chapter 2, we will use the notations

$$\Omega_i := \operatorname{supp}(\varphi_i), \qquad \Omega_t := \bigcup_{i\in\hat{t}} \Omega_i \tag{5.5}$$

in the following.

Lemma 5.1 *Let $\mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, k) \in \mathbb{R}^{\mathcal{I}\times\mathcal{I}}$ be an arbitrary $\mathcal{H}$-matrix format, where each admissible block $(t,s) \in T_{\mathcal{I}\times\mathcal{I}}$ satisfies at least the simple admissibility condition*

$$\min\{\operatorname{diam}(\Omega_t), \operatorname{diam}(\Omega_s)\} \leq \eta \operatorname{dist}(\Omega_t, \Omega_s)$$

with $\eta > 0$. Then any finite-element matrix belongs (exactly) to $\mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, k)$.

Proof: Let $b = (t,s)$ be an admissible block. Because of $\eta > 0$, $\operatorname{dist}(\Omega_t, \Omega_s) > 0$ follows. Therefore, for all $i \in \hat{t}$, $j \in \hat{s}$ the supports of φ_i and φ_j are disjoint, which implies $A_{ij} = 0$. This shows $A|_{\hat{t}\times\hat{s}} = 0$. Hence, the local rank is 0 which is $\leq k$ for any k from $\mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, k)$. ∎

5.2 The Mass Matrix

The mass matrix (or Gram matrix) $M \in \mathbb{R}^{\mathcal{I}\times\mathcal{I}}$ defined by

$$M_{ij} = \int_\Omega \varphi_i(x)\varphi_j(x)\mathrm{d}x \qquad \text{for all } i,j \in \mathcal{I}$$

is the finite-element approximation of the identity with respect to the finite-element space V_n from above.

We introduce the prolongation

$$P : \mathbb{R}^{\mathcal{I}} \to V_n, \qquad \mathbf{v} = (v_j)_{j\in\mathcal{I}} \mapsto \sum_{j\in\mathcal{I}} v_j\varphi_j, \tag{5.6a}$$

and the restriction

$$R : V_n \to \mathbb{R}^{\mathcal{I}}, \qquad v \mapsto \mathbf{v} = \left(\int_\Omega \varphi_i(x) v(x) \, \mathrm{d}x \right)_{i \in \mathcal{I}}. \tag{5.6b}$$

We can see that $R = P^*$ holds, i.e., that R is the L^2-adjoint of P.

Exercise 5 *Show that a) $M = RP$, b) M is positive definite, and c) the extreme eigenvalues $\mu_{\min}$ and $\mu_{\max}$ of M are the best bounds in the inequalities*

$$\sqrt{\mu_{\min}} \|\mathbf{v}\|_2 \leq \|P\mathbf{v}\|_{L^2} \leq \sqrt{\mu_{\max}} \|\mathbf{v}\|_2 \qquad \text{for all } \mathbf{v} \in \mathbb{R}^{\mathcal{I}}. \tag{5.6c}$$

If the triangles $\tau \in \mathcal{T}$ have comparable size, i.e., if

$$\frac{\operatorname{diam}(\tau)}{\operatorname{diam}(\tau')} \leq C_q \qquad \text{holds for all } \tau, \tau' \in \mathcal{T},$$

the triangulation $\mathcal{T}$ is called *quasi-uniform*.

The triangulation $\mathcal{T}$ is called *form-regular*, if all angles of the triangles $\tau \in \mathcal{T}$ are bounded from below by a fixed angle $\alpha_0 > 0$.

Lemma 5.2 *If $\mathcal{T}$ is quasi-uniform and form-regular, the norms given by $\|P\mathbf{v}\|_{L^2}/\operatorname{vol}(\Omega)$ and $\|\mathbf{v}\|_2/\sqrt{n}$ for all $\mathbf{v} \in \mathbb{R}^{\mathcal{I}}$ are equivalent with a constant independent of $n = \#\mathcal{I}$.*

In particular, the condition $\operatorname{cond}(M) = \mu_{\max}/\mu_{\min}$ is independent of n.

Proof: see [38, Remark 8.8.4]. ∎

The norm defined by

$$\|\mathbf{v}\|_{V_n} := \|P\mathbf{v}\|_{L^2} \qquad \text{for all } \mathbf{v} \in \mathbb{R}^{\mathcal{I}}$$

has the major advantage that it does not depend on the choice of basis functions and that refining the grid will not change its scaling. Using the mass matrix, we can write the norm as

$$\|\mathbf{v}\|_{V_n} = \|P\mathbf{v}\|_{L^2} = \sqrt{\langle P\mathbf{v}, P\mathbf{v}\rangle_{L^2}} = \sqrt{\langle M\mathbf{v}, \mathbf{v}\rangle_2} = \|M^{1/2}\mathbf{v}\|_2.$$

We require a counterpart of this norm for functionals. The natural choice is the dual norm

$$\|\mathbf{f}\|_{V_n'} := \sup_{\mathbf{v}\neq 0} \frac{\langle \mathbf{f}, \mathbf{v}\rangle_2}{\|P\mathbf{v}\|_{L^2}} = \sup_{\mathbf{v}\neq 0} \frac{\langle \mathbf{f}, \mathbf{v}\rangle_2}{\|M^{1/2}\mathbf{v}\|_2} = \sup_{\mathbf{v}\neq 0} \frac{\langle \mathbf{f}, M^{-1/2}\mathbf{v}\rangle_2}{\|\mathbf{v}\|_2} = \|M^{-1/2}\mathbf{f}\|_2.$$

These vector norms give rise to matrix norms. We define

$$\|X\|_{V_n' \leftarrow V_n} := \sup_{\mathbf{v}\neq 0} \frac{\|X\mathbf{v}\|_{V_n'}}{\|\mathbf{v}\|_{V_n}} = \|M^{-1/2} X M^{-1/2}\|_2 \qquad \text{for all } X \in \mathbb{R}^{\mathcal{I}\times\mathcal{I}},$$

$$\|X\|_{V_n \leftarrow V_n'} := \sup_{\mathbf{v}\neq 0} \frac{\|X\mathbf{v}\|_{V_n}}{\|\mathbf{v}\|_{V_n'}} = \|M^{1/2} X M^{1/2}\|_2 \qquad \text{for all } X \in \mathbb{R}^{\mathcal{I}\times\mathcal{I}}.$$

We will frequently use the first of these two norms, so we introduce the short notation

$$|||X||| := \|X\|_{V_n' \leftarrow V_n} \qquad \text{for all } X \in \mathbb{R}^{\mathcal{I}\times\mathcal{I}}$$

and observe that

$$|||X||| = \|M^{-1/2} X M^{-1/2}\|_2 = \|P M^{-1} X M^{-1} R\|_{L^2 \leftarrow L^2} \qquad \text{holds for all } X \in \mathbb{R}^{\mathcal{I}\times\mathcal{I}}. \tag{5.7}$$

Let us consider the application of the norm $|||\cdot|||$ to the matrices corresponding to integral operators introduced in the Chapters 1 and 3: for a kernel function $g : \Omega \times \Omega \to \mathbb{R}$, we define the operator $\mathcal{G}$ by

$$\mathcal{G}[u](x) := \int_\Omega g(x,y)u(y)\,\mathrm{d}y \qquad \text{for all } u \in V_n \text{ and all} x \in \Omega.$$

Its discrete counterpart is the matrix $G \in \mathbb{R}^{\mathcal{I}\times\mathcal{I}}$ given by

$$G_{ij} = \int_\Omega \varphi_i(x)\mathcal{G}[\varphi_j](x)\,\mathrm{d}x = \int_\Omega \varphi_i(x)\int_\Omega g(x,y)\varphi_j(y)\,\mathrm{d}y\,\mathrm{d}x \qquad \text{for all } i,j \in \mathcal{I}.$$

Remark 5.3 *Let $C_{\mathrm{fe}} := \mu_{\max}/\mu_{\min}$. For all $\mathbf{u} = (u_i)_{i\in\mathcal{I}}$, we have*

$$\left\|\sum_{i\in\mathcal{I}} u_i\varphi_i\right\|_{L^2}^2 \leq C_{\mathrm{fe}}\sum_{i\in\mathcal{I}}\|u_i\varphi_i\|_{L^2}^2 \leq C_{\mathrm{fe}}^2\left\|\sum_{i\in\mathcal{I}} u_i\varphi_i\right\|_{L^2}^2. \tag{5.8}$$

Proof: Let $\mathbf{u} \in \mathbb{R}^{\mathcal{I}}$. We define $\mathbf{e}_i := (\delta_{ij})_{j\in\mathcal{I}}$ for all $i \in \mathcal{I}$ and get

$$\begin{aligned}\left\|\sum_{i\in\mathcal{I}} u_i\varphi_i\right\|_{L^2}^2 &= \|P\mathbf{u}\|_{L^2}^2 \leq \mu_{\max}\|\mathbf{u}\|_2^2 = \mu_{\max}\sum_{i\in\mathcal{I}}\|u_i\mathbf{e}_i\|_2^2 \leq \frac{\mu_{\max}}{\mu_{\min}}\sum_{i\in\mathcal{I}}\|u_i\varphi_i\|_{L^2}^2\\ &\leq \frac{\mu_{\max}^2}{\mu_{\min}}\sum_{i\in\mathcal{I}}\|u_i\mathbf{e}_i\|_2^2 = \frac{\mu_{\max}^2}{\mu_{\min}}\|\mathbf{u}\|_2^2 \leq \frac{\mu_{\max}^2}{\mu_{\min}^2}\|P\mathbf{u}\|_{L^2}^2 = \frac{\mu_{\max}^2}{\mu_{\min}^2}\left\|\sum_{i\in\mathcal{I}} u_i\varphi_i\right\|_{L^2}^2.\end{aligned}$$

■

We can use this estimate in order to prove an equivalent of Theorem 3.7 for the norm $|||\cdot|||$:

Theorem 5.4 *Let $\epsilon \in \mathbb{R}_{\geq 0}$, and let C_{sp}^* be defined as in (3.16). Let $(\mathcal{G}_{t,s})_{b=(t,s)\in\mathcal{L}(T_{\mathcal{I}\times\mathcal{I}})}$ be a family of operators satisfying*

$$\|\mathcal{G}_{t,s}\|_{L^2(\Omega_t)\leftarrow L^2(\Omega_s)} \leq \epsilon \qquad \text{for all } b = (t,s) \in \mathcal{L}(T_{\mathcal{I}\times\mathcal{I}}).$$

Let $p \in \mathbb{N}_0$ be the depth of the cluster tree $T_{\mathcal{I}}$. Let the matrix $\tilde{G} \in \mathbb{R}^{\mathcal{I}\times\mathcal{I}}$ be defined by

$$G_{ij} := \langle\varphi_i, \mathcal{G}_{t,s}[\varphi_j]\rangle_{L^2} \qquad \text{for } i \in \hat{t}, j \in \hat{s}, b = (t,s) \in \mathcal{L}(T_{\mathcal{I}\times\mathcal{I}}).$$

Then we have

$$|||G||| \leq C_{\mathrm{fe}}^2 C_{\mathrm{sp}}^*(p+1)\epsilon.$$

Proof: Let $\mathbf{u}, \mathbf{v} \in \mathbb{R}^{\mathcal{I}}$. Due to (2.2), we find

$$\begin{aligned}\langle\mathbf{v}, G\mathbf{u}\rangle_2 &= \sum_{(t,s)\in\mathcal{L}(T_{\mathcal{I}\times\mathcal{I}})}\sum_{i\in\hat{t}}\sum_{j\in\hat{s}} v_iu_jG_{ij}\\ &= \sum_{(t,s)\in\mathcal{L}(T_{\mathcal{I}\times\mathcal{I}})}\sum_{i\in\hat{t}}\sum_{j\in\hat{s}} v_iu_j\langle\varphi_i, \mathcal{G}_{t,s}[\varphi_j]\rangle_{L^2}\\ &= \sum_{(t,s)\in\mathcal{L}(T_{\mathcal{I}\times\mathcal{I}})}\left\langle\sum_{i\in\hat{t}} u_i\varphi_i, \mathcal{G}_{t,s}\left[\sum_{j\in\hat{s}} v_j\varphi_j\right]\right\rangle\\ &\leq \sum_{(t,s)\in\mathcal{L}(T_{\mathcal{I}\times\mathcal{I}})}\left\|\sum_{i\in\hat{t}} u_i\varphi_i\right\|_{L^2(\Omega_t)}\left\|\mathcal{G}_{t,s}\left[\sum_{j\in\hat{s}} v_j\varphi_j\right]\right\|_{L^2(\Omega_t)}\end{aligned}$$

$$\leq \epsilon \sum_{(t,s)\in\mathcal{L}(T_{\mathcal{I}\times\mathcal{I}})} \left\|\sum_{i\in\hat{t}} u_i\varphi_i\right\|_{L^2(\Omega_t)} \left\|\sum_{j\in\hat{s}} v_j\varphi_j\right\|_{L^2(\Omega_s)}$$

$$\leq \epsilon \left(\sum_{(t,s)\in\mathcal{L}(T_{\mathcal{I}\times\mathcal{I}})} \left\|\sum_{i\in\hat{t}} u_i\varphi_i\right\|^2_{L^2(\Omega)}\right)^{1/2} \left(\sum_{(t,s)\in\mathcal{L}(T_{\mathcal{I}\times\mathcal{I}})} \left\|\sum_{j\in\hat{s}} v_j\varphi_j\right\|^2_{L^2(\Omega)}\right)^{1/2}.$$

Now we can use the L^2-stability (5.8) of the basis and Lemma 2.7 to prove

$$\begin{aligned}
\sum_{(t,s)\in\mathcal{L}(T_{\mathcal{I}\times\mathcal{I}})} \left\|\sum_{i\in\hat{t}} u_i\varphi_i\right\|^2_{L^2(\Omega)} &\leq C_{\mathrm{sp}}^* \sum_{t\in T_{\mathcal{I}}} \left\|\sum_{i\in\hat{t}} u_i\varphi_i\right\|^2_{L^2(\Omega)} = C_{\mathrm{sp}}^* \sum_{\ell=0}^{p} \sum_{t\in T_{\mathcal{I}}^{(\ell)}} \left\|\sum_{i\in\hat{t}} u_i\varphi_i\right\|^2_{L^2(\Omega)} \\
&\leq C_{\mathrm{fe}} C_{\mathrm{sp}}^* \sum_{\ell=0}^{p} \sum_{t\in T_{\mathcal{I}}^{(\ell)}} \sum_{i\in\hat{t}} \|u_i\varphi_i\|^2_{L^2(\Omega)} \leq C_{\mathrm{fe}} C_{\mathrm{sp}}^* \sum_{\ell=0}^{p} \sum_{i\in\mathcal{I}} \|u_i\varphi_i\|^2_{L^2(\Omega)} \\
&\leq C_{\mathrm{fe}}^2 C_{\mathrm{sp}}^* \sum_{\ell=0}^{p} \left\|\sum_{i\in\mathcal{I}} u_i\varphi_i\right\|^2_{L^2(\Omega)} = C_{\mathrm{fe}}^2 C_{\mathrm{sp}}^* (p+1) \|P\mathbf{u}\|^2_{L^2(\Omega)} \\
&= C_{\mathrm{fe}}^2 C_{\mathrm{sp}}^* (p+1) \|\mathbf{u}\|^2_{V_n}.
\end{aligned}$$

This means that we have

$$\langle \mathbf{v}, G\mathbf{u}\rangle_2 \leq C_{\mathrm{fe}}^2 C_{\mathrm{sp}}^* (p+1)\epsilon \|\mathbf{u}\|_{V_n} \|\mathbf{v}\|_{V_n}$$

and can conclude

$$|||G||| = \|G\|_{V_n' \leftarrow V_n} = \sup_{\mathbf{u},\mathbf{v}\neq 0} \frac{\langle \mathbf{v}, G\mathbf{u}\rangle}{\|\mathbf{v}\|_{V_n}\|\mathbf{u}\|_{V_n}} \leq C_{\mathrm{fe}}^2 C_{\mathrm{sp}}^* (p+1)\epsilon.$$

■

5.3 $\mathcal{H}$-Matrix Approximation of the Inverse of the Mass Matrix

Our construction of the inverse of the stiffness matrix A of the discrete problem (5.3) employs the inverse of the mass matrix M to switch between the discrete space and its dual. Before we can prove that A can be approximated by an $\mathcal{H}$-matrix, we have to establish that we can approximate M in this format.

The analysis is based on the *graph* $G(M)$ of a matrix M (cf. [39, §6.2]): the nodes of the graph are the indices $i \in \mathcal{I}$, and for $i, j \in \mathcal{I}$, (i, j) is a (directed) edge in $G(M)$ if and only if $M_{ij} \neq 0$ holds. A *path* from $i \in \mathcal{I}$ to $j \in \mathcal{I}$ is a sequence $(i_\iota)_{\iota=0}^m$ of nodes $i_\iota \in \mathcal{I}$ such that $(i_{\iota-1}, i_\iota)$ is an edge in $G(M)$ for all $\iota \in \{1, \ldots, m\}$. In this context, the number $m \in \mathbb{N}_0$ is called the *length* of the path. The *graph distance* $\delta(i, j)$ is defined to be the length of the shortest path from i to j in $G(M)$.

We assume that the discrete space V_n is defined based on a conforming triangulation $\mathcal{T}$ with nodes $(x_i)_{i\in\mathcal{I}}$ and let $h \in \mathbb{R}_{>0}$ denote the maximal length of an edge of $\mathcal{T}$, i.e.,

$$h := \max\{\operatorname{diam}(\tau) \ : \ \tau \in \mathcal{T}\}. \tag{5.9}$$

Remark 5.5 *a) For all $i \in \mathcal{I}$, h is a bound of the radius of the support Ω_i with respect to the center x_i:*

$$\max\{\|x_i - x\|_2 \ : \ x \in \Omega_i\} = \operatorname{dist}(x_i, \mathbb{R}^d \setminus \Omega_i) \leq h.$$

b) Let $i, j \in \mathcal{I}$, and let $(i_\iota)_{\iota=0}^m$ be a path from i to j in $G(M)$ of minimal length $\delta := \delta(i, j) \in \mathbb{N}_0$. Then $\|x_i - x_j\|_2 \leq \delta h$ holds.

c) Furthermore,

$$\delta \geq 2 + \operatorname{dist}(\Omega_i, \Omega_j)/h$$

holds if $\operatorname{dist}(\Omega_i, \Omega_j) > 0$ or $\Omega_i \cap \Omega_j$ is of measure zero. All $i \neq j$ satisfy $\delta \geq 1 + \operatorname{dist}(\Omega_i, \Omega_j)/h$.

Proof: a) The set $\{\|x_i - x\|_2 : x \in \Omega_i\}$ takes its maxima at the corner points. For each corner point x' of Ω_i there is a triangle $\tau \in \mathcal{T}$, so that the line $\overline{x_i\, x'}$ coincides with a side of t. Hence $\|x_i - x'\|_2 \leq h$.

b) Let (i_m, i_{m+1}) be an edge in $G(M)$, i.e., $M_{i_m,i_{m+1}} \neq 0$. Hence the supports Ω_{i_m} and $\Omega_{i_{m+1}}$ must overlap in an inner point. This case happens only if the nodal point $x_{i_{m+1}}$ is one of the corner points of Ω_{i_m} or if it coincides with x_{i_m} itself. Hence, $\|x_{i_m} - x_{i_{m+1}}\|_2 \leq h$. A simple induction yields $\|x_i - x_j\|_2 \leq \delta h$.

c) Let $\delta \geq 2$. Because of $x_{i_1} \in \Omega_i$ and $x_{i_{\delta-1}} \in \Omega_j$, $\operatorname{dist}(\Omega_i, \Omega_j) \leq \|x_{i_1} - x_{i_{\delta-1}}\|_2 \leq \delta(i_1, i_{\delta-1})h$ holds. Due to the minimality of δ, we have $\delta(i_1, i_{\delta-1}) = \delta - 2$ and conclude $\operatorname{dist}(\Omega_i, \Omega_j) \leq (\delta - 2)h$, which implies the further assertions. ∎

The mass matrix is positive definite and therefore the norms and the extreme eigenvalues are connected by $\mu_{\min} = \|M^{-1}\|_2^{-1}$ and $\mu_{\max} = \|M\|_2$.

Let $T_{\mathcal{I}\times\mathcal{I}}$ be a level-consistent block cluster tree satisfying the admissibility condition

$$\max\{\operatorname{diam}(\Omega_t), \operatorname{diam}(\Omega_s)\} \leq \eta \operatorname{dist}(\Omega_t, \Omega_s) \qquad \text{for all admissible blocks } b = (t, s) \in \mathcal{L}^+(T_{\mathcal{I}\times\mathcal{I}}) \tag{5.10}$$

(here the weaker condition $\sqrt{\operatorname{diam}(\Omega_t)\operatorname{diam}(\Omega_s)}) \leq \eta \operatorname{dist}(\Omega_t, \Omega_s)$ would be sufficient).

Criterion 1 *Assume*

$$\operatorname{diam}(\Omega_t)^d \geq C_1 h^d \#\hat{t} \qquad \text{for all } t \in T_{\mathcal{I}} \tag{5.11a}$$

with certain constants C_1, d, and $h > 0$ from (5.9). Let $M \in \mathbb{R}^{I\times I}$ be a matrix with the entry-wise estimate

$$|M_{ij}| \leq c_2 q^{\operatorname{dist}(\Omega_i,\Omega_j)/h} \qquad \text{for all } i, j \in \mathcal{I}, \text{ where } q < 1. \tag{5.11b}$$

Then the blockwise definition

$$M_k|_{\hat{t}\times\hat{s}} := \begin{cases} M|_{\hat{t}\times\hat{s}} & \text{for } b = (t, s) \in \mathcal{L}(T_{\mathcal{I}\times\mathcal{I}}) \text{ with } (\#\hat{t})(\#\hat{s}) \leq k^2, \\ 0 & \text{for } b = (t, s) \in \mathcal{L}(T_{\mathcal{I}\times\mathcal{I}}) \text{ with } (\#\hat{t})(\#\hat{s}) > k^2, \end{cases} \tag{5.11c}$$

leads to $M_k \in \mathcal{H}(k, P)$ with the error estimate

$$\|M - M_k\|_2 \leq C_{\mathrm{sp}}^* c_2 (1 + \operatorname{depth}(T_{\mathcal{I}\times\mathcal{I}})) \cdot \mathcal{O}(r^{k^{1/d}}) \qquad \text{for any } r < q^{C_1/\eta} < 1, \tag{5.11d}$$

where C_{sp}^* is defined as in (3.16).

Proof: a) Since $\operatorname{rank}(M_k|_b) \leq \min\{\#\hat{t}, \#\hat{s}\} \leq \sqrt{(\#\hat{t})(\#\hat{s})} \leq k$ for all $b = (t, s) \in \mathcal{L}(T_{\mathcal{I}\times\mathcal{I}})$, M_k belongs to $\mathcal{H}(k, P)$. Let $E = M - M_k$ be the error matrix. $E|_{\hat{t}\times\hat{s}} = 0$ holds, if $(\#\hat{t})(\#\hat{s}) \leq k^2$. Theorem 3.7 provides an estimate of $\|E\|_2$ by means of the local spectral norms of $E|_b = M|_b$ which are to be considered for the remaining case of $\#\tau\#\sigma > k^2$.

b) For $(\#\hat{t})(\#\hat{s}) > k^2$ and $i \in \hat{t}$, $j \in \hat{s}$ we have to estimate the error component $E_{ij} = M_{ij}$. The admissibility condition implies

$$\frac{\operatorname{dist}(\Omega_i, \Omega_j)}{h} \underset{i\in\hat{t}, j\in\hat{s}}{\geq} \frac{\operatorname{dist}(\Omega_t, \Omega_s)}{h} \underset{(5.10)}{\geq} \frac{\sqrt{\operatorname{diam}(\Omega_t)\operatorname{diam}(\Omega_s)}}{h\eta} \underset{(5.11a)}{\geq} \frac{C_1}{\eta} \sqrt[2d]{(\#\hat{t})(\#\hat{s})}.$$

From inequality (5.11b) we derive

$$|E_{ij}| = |M_{ij}| \leq c_2 q^{C_1 \sqrt[2d]{(\#\hat{t})(\#\hat{s})}/\eta}.$$

A rough estimate of the spectral norm is provided by

$$\|E|_{\hat{t}\times\hat{s}}\|_2 \leq \sqrt{(\#\hat{t})(\#\hat{s})} \max_{i\in\hat{t}, j\in\hat{s}} |E_{ij}| \leq c_2 \sqrt{(\#\hat{t})(\#\hat{s})} q^{C_1 \sqrt[2d]{(\#\hat{t})(\#\hat{s})}/\eta}.$$

The right-hand side can be simplified: Let $k_{\min} \le k$. There are constants $c_1' = c_1'(k_{\min}) > C_1$ and c_2'' such that $\ell\, q^{C_1\sqrt[2d]{\ell}/\eta} \le c_2'' q^{c_1'\sqrt[2d]{\ell}/\eta}$ for all $\ell > k_{\min}$. Hence, with $c_2' := c_2 c_2''$, we have

$$\|E|_b\|_2 \le c_2\, c_2''\, q^{c_1'\sqrt[2d]{\#\tau\#\sigma}/\eta} \underset{\#\tau\#\sigma > k^2}{<} c_2'\, q^{c_1'\sqrt[d]{k}/\eta}.$$

Lemma 3.7 shows $\|E\|_2 \le C_{\mathrm{sp}}^*(1+\mathrm{depth}(T_{\mathcal{I}\times\mathcal{I}}))c_2' q^{c_1'\sqrt[d]{k}/\eta}$. The inequalities $c_1' > C_1$ and $r := q^{c_1'/\eta} < q^{C_1/\eta}$ are equivalent and yield (5.11d). ∎

In the following we have to satisfy the conditions (5.11a,b) of the criterion. The first step is done by the next lemma which applies to all positive definite, well-conditioned matrices. Note that the optimal values of a, b in the next lemma are the extreme eigenvalues resulting in $r = \mathrm{cond}(M)$.

Lemma 5.6 *Let $M \in \mathbb{R}^{\mathcal{I}\times\mathcal{I}}$ be a symmetric positive definite matrix with spectrum $\sigma(M) \subset [a,b]$, where $0 < a \le b$. For $i, j \in \mathcal{I}$ denote the graph distance of i, j by $\delta(i,j)$. Then for all $i \ne j$, the following estimate holds:*

$$|(M^{-1})_{ij}| \le \hat{c}\, q^{\delta(i,j)} \qquad \text{with } \hat{c} = \frac{(1+\sqrt{r})^2}{2ar},\ q = \frac{\sqrt{r}-1}{\sqrt{r}+1}, \quad \text{where } r := \frac{b}{a}. \tag{5.12}$$

Proof: For any polynomial $p \in \mathcal{P}_k$ with $k < \delta_{ij}$ we observe that $p(M)_{ij} = 0$ holds. Furthermore, the spectral norm and the spectral radius coincide for normal matrices:

$$\|M^{-1} - p(M)\|_2 = \rho(M^{-1} - p(M)) = \max_{\mu\in\sigma(M)} |\mu^{-1} - p(\mu)|.$$

Due to a result of Chebyshev (cf. [49, p. 33]) there is a polynomial $p_k \in \mathcal{P}_k$ so that

$$\|\mu^{-1} - p_k(\mu)\|_{\infty,[a,b]} \le \hat{c}\, q^{k+1}$$

with q and $\hat{c}$ as in (5.12). Set $k := \delta_{ij} - 1$. Then

$$|(M^{-1})_{ij}| = |(M^{-1})_{ij} - p_k(M)_{ij}| \le \|M^{-1} - p_k(M)\|_2 \le \hat{c}\, q^{k+1} = \hat{c}\, q^{\delta_{ij}}$$

proves the assertion. ∎

Since the exponent in (5.12) satisfies $\delta(i,j) \ge 1 + \mathrm{dist}(\Omega_i, \Omega_j)/h$ due to Remark 5.5b, the condition (5.11a) holds: $\hat{c}\, q^{\delta(i,j)} \le \hat{c} q^{1+\mathrm{dist}(\Omega_i,\Omega_j)/h} = (\hat{c}q)\, q^{\mathrm{dist}(\Omega_i,\Omega_j)/h} = c_2 q^{\mathrm{dist}(\Omega_i,\Omega_j)/h}$ with

$$c_2 = \hat{c}q = C\,\|M^{-1}\|_2, \qquad C := \frac{r-1}{2r}, \tag{5.13}$$

provided that $a = \mu_{\min}$ and $b = \mu_{\max}$ are the extreme eigenvalues of M.

Under the assumption of shape regularity and quasi-uniformity, we have the estimate

$$\mathrm{vol}(\Omega_i) \ge c_v h^d. \tag{5.14}$$

The supports Ω_i may overlap, but due to the form regularity the number of intersecting Ω_i is limited. This fact is expressed in the following inequality: there is a constant $C_{\mathrm{ov}} > 0$ so that

$$C_{\mathrm{ov}}\,\mathrm{vol}(\Omega_t) \ge \sum_{i\in\hat{t}} \mathrm{vol}(\Omega_i). \tag{5.15}$$

Lemma 5.7 *The inequalities (5.14) and (5.15) imply the condition (5.11a), i.e.,*

$$\mathrm{diam}(\Omega_t)^d \ge C_1 h^d \#\hat{t} \qquad \text{with } C_1 := \frac{c_v}{\omega_d C_{\mathrm{ov}}},$$

where ω_d is the volume of the d-dimensional unit sphere.

Proof: Ω_t is contained in the sphere of radius $\operatorname{diam}(\Omega_t)$, so that $\operatorname{diam}(\Omega_t)^d \geq \operatorname{vol}(\Omega_t)/\omega_d$. On the other hand, (5.15) and (5.14) show that $\operatorname{vol}(\Omega_t) \geq \frac{1}{C_{\mathrm{ov}}} \sum_{i \in \hat{t}} \operatorname{vol}(\Omega_i) \geq \frac{c_v}{C_{\mathrm{ov}}} h^d \# \hat{t}$. ∎

Theorem 5.8 *Assume that the admissible leaves $b = (t, s) \in \mathcal{L}^+(T_{\mathcal{I}\times\mathcal{I}})$ of $T_{\mathcal{I}\times\mathcal{I}}$ satisfy the admissibility condition (5.10). Assume form regular and quasi-uniform finite elements (in particular (5.14), (5.15) and $\operatorname{cond}(M) = \mathcal{O}(1)$ are needed; cf. Lemmata 5.2, 5.7). Then for all $\varepsilon > 0$ there is a matrix $N_{\mathcal{H}} \in \mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, k_\varepsilon)$ such that*

$$\|M^{-1} - N_{\mathcal{H}}\|_2 \leq \varepsilon \|M^{-1}\|_2 \qquad \text{with } k_\varepsilon = \mathcal{O}(\log^d((p+1)/\varepsilon) \text{ and } p := \operatorname{depth}(T_{\mathcal{I}\times\mathcal{I}}). \tag{5.16}$$

Proof: In Criterion 1 (with M replaced by M^{-1}) the conditions (5.11a,b) are satisfied with the constants $C_1 := \frac{c_v}{\omega_d C_{\mathrm{ov}}}$ (see Lemma 5.7) and $c_2 := C \|M^{-1}\|_2$ from (5.13). The error estimate (5.11d) yields

$$\|M^{-1} - N_{\mathcal{H}}\|_2 \leq \|M^{-1}\|_2 \cdot C^*_{\mathrm{sp}}(1 + \operatorname{depth}(T_{\mathcal{I}\times\mathcal{I}})) \cdot \mathrm{const} \cdot \rho^{k^{1/d}} \tag{5.17}$$

with quantities C^*_{sp}, const and $\rho < q^{C_1/\eta}$ (η from (5.10)) independent of $\#\mathcal{I}$. Hence, (5.16) follows for $k = k_\varepsilon = \mathcal{O}(\log^d((p+1)/\varepsilon))$ with $p = \operatorname{depth}(T_{\mathcal{I}\times\mathcal{I}})$. ∎

Inequality (5.16) describes the relative error with respect to the spectral norm. Later, the norm $|||A|||$ from (5.7) will be of interest.

Corollary 5.9 *Under the assumptions of Theorem 5.8 the inequality*

$$|||M(M^{-1} - N_{\mathcal{H}})M||| = \|M^{1/2}(M^{-1} - N_{\mathcal{H}})M^{1/2}\|_2 = \|P(M^{-1} - N_{\mathcal{H}})R\|_{L^2(\Omega)\leftarrow L^2(\Omega)} \tag{5.18a}$$

$$\leq C^*_{\mathrm{sp}}(1 + \operatorname{depth}(T_{\mathcal{I}\times\mathcal{I}})) \cdot \mathrm{const}' \cdot \rho^{k^{1/d}}, \tag{5.18b}$$

holds, where const′ := const · cond(M) with const from (5.17). As in Theorem 5.8, for all $\varepsilon > 0$ there exists a matrix $N_{\mathcal{H}} \in \mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, k_\varepsilon))$, such that

$$\|M^{1/2}(M^{-1} - N_{\mathcal{H}})M^{1/2}\|_2 \leq \varepsilon \qquad \text{with } k_\varepsilon = \mathcal{O}(\log^d((1 + \operatorname{depth}(T_{\mathcal{I}\times\mathcal{I}}))/\varepsilon)). \tag{5.18c}$$

Proof: For (5.18a) compare (5.7). Furthermore,

$$\begin{aligned}\|M^{1/2}(M^{-1} - N_{\mathcal{H}})M^{1/2}\|_2 &\leq \|M^{1/2}\|_2 \|M^{-1} - N_{\mathcal{H}}\|_2 \|M^{1/2}\|_2 \\ &= \|M^{-1} - N_{\mathcal{H}}\|_2 \|M^{1/2}\|_2^2 = \|M^{-1} - N_{\mathcal{H}}\|_2 \|M\|_2.\end{aligned}$$

Inequality (5.17) shows

$$\|M^{1/2}(M^{-1} - N_{\mathcal{H}})M^{1/2}\|_2 \leq \|M^{-1}\|_2 \|M\|_2 \cdot C^*_{\mathrm{sp}}(1 + \operatorname{depth}(T_{\mathcal{I}\times\mathcal{I}})) \cdot \mathrm{const} \cdot \rho^{k^{1/d}}.$$

Since $\|M^{-1}\|_2 \|M\|_2 = \operatorname{cond}(M) = \mathcal{O}(1)$, the assertion is proved. ∎

5.4 The Green Function Operator and its Galerkin Discretisation

5.4.1 The Elliptic Boundary Value Problem

We consider the differential operator

$$Lu = -\operatorname{div}(C(x) \operatorname{grad} u) \qquad \text{in } \Omega \tag{5.19}$$

from (5.1), where $\Omega \subset \mathbb{R}^d$ is assumed to be a bounded Lipschitz domain.

The Dirichlet boundary value problem is

$$\begin{array}{rcll} Lu & = & f & \text{in } \Omega, \\ u & = & 0 & \text{on } \Gamma := \partial\Omega. \end{array} \tag{5.20}$$

We assume *no smoothness* of the $d \times d$ coefficient matrix $C(x)$ except its boundedness $C \in L^\infty(\Omega)$. The uniform ellipticity is described by the inequalities

$$0 < \lambda_{\min} \le \lambda \le \lambda_{\max} \qquad \text{for all eigenvalues } \lambda \in \sigma(C(x)) \text{ for almost all } x \in \Omega. \tag{5.21}$$

The ratio

$$\kappa_C = \lambda_{\max}/\lambda_{\min} \tag{5.22}$$

is an upper bound of all spectral condition numbers $\operatorname{cond}_2 C(x)$. But notice that the extrema $\lambda_{\min}$ and $\lambda_{\max}$ need not be attained for the same $x \in \Omega$.

After fixing the basis of $V_n \subset V = H_0^1(\Omega)$, the finite-element matrix A is defined by $A_{ij} = a(\phi_j, \phi_i)$ using the weak formulation (5.2) (see (5.4)).

Using P, R from (5.6a,b) and $L : H_0^1(\Omega) \to H^{-1}(\Omega)$ from $a(u,v) = (Lu, v)_{L^2(\Omega)}$, a possible representation of the stiffness matrix A is

$$A = RLP. \tag{5.23}$$

The following statements also hold, when L contains further terms of first and zeroth order with L^∞-coefficients (cf. Bebendorf [5]). The case of dominating convection is discussed in Bebendorf [6]. Elliptic systems of equations (e.g., the Lamé equation) are considered in [57].

5.4.2 The Green Function

For all $x, y \in \Omega$ the Green function $G(x,y)$ is the solution of $LG(\cdot, y) = \delta_y$ with boundary data $G(\cdot, y)|_\Gamma = 0$ (L and the restriction to Γ refer to the first variable $\cdot$), where δ_y is the Dirac distribution at $y \in \Omega$. The Green function is the Schwartz kernel of the inverse L^{-1}, i.e., the solution of (5.20) is represented by

$$u(x) = \int_\Omega G(x,y)\, f(y)\, \mathrm{d}y.$$

For $L = -\Delta$ (i.e., $C(x) = I$) the Green function is analytic in Ω. Since, under the present assumptions the coefficient matrix $C(x)$ is only bounded, G is not necessarily differentiable. The existence of Green's function is proved for $d \ge 3$ by Grüter-Widman [36] together with the estimate

$$|G(x,y)| \le \frac{C_G}{\lambda_{\min}} |x-y|^{2-d} \qquad (C_G = C_G(\kappa_C) \text{ with } \kappa_C \text{ from (5.22)}, \lambda_{\min} \text{ from (5.21)}). \tag{5.24a}$$

For $d = 2$, the existence proof can be found in Dolzmann-Müller [20] together with

$$|G(x,y)| \le \frac{C_G}{\lambda_{\min}} \log |x-y|. \tag{5.24b}$$

Here, $|\cdot|$ denotes the Euclidean norm in $\mathbb{R}^d$.

5.4.3 The Green Function Operator $\mathcal{G}$

Because of (5.24a,b) the integral operator

$$\left(L^{-1} f\right)(x) = (\mathcal{G} f)(x) := \int_\Omega G(x,y) f(y) \mathrm{d}y \qquad (x \in \Omega) \tag{5.25}$$

is well-defined. Usually, G is not explicitly known, however, we will use $\mathcal{G}$ only for theoretical considerations.

Lemma 5.10 *Under the mentioned assumptions* $\mathcal{G} \in \mathcal{L}\left(L^2(\Omega), L^2(\Omega)\right)$. *More precisely, the bound is*

$$\|\mathcal{G}\|_{L^2(\Omega) \leftarrow L^2(\Omega)} \le \operatorname{diam}(\Omega)^2 / \lambda_{\min}. \tag{5.26}$$

Proof: Let $u = \mathcal{G}f \in H_0^1(\Omega)$ with $f \in L^2(\Omega)$. (5.21) shows $a(u,u) \geq \lambda_{\min}\|\nabla u\|^2_{L^2(\Omega)}$. The estimate $\|u\|^2_{L^2(\Omega)} \leq \operatorname{diam}(\Omega)^2\|\nabla u\|^2_{L^2(\Omega)}$ holds for a bounded domain Ω and functions $u \in H_0^1(\Omega)$ so that

$$\|u\|^2_{L^2(\Omega)} \leq \operatorname{diam}(\Omega)^2\|\nabla u\|^2_{L^2(\Omega)} \leq \frac{\operatorname{diam}(\Omega)^2}{\lambda_{\min}} a(u,u).$$

On the other hand, $a(u,u) = (f,u)_{L^2(\Omega)} \leq \|f\|_{L^2(\Omega)}\|u\|_{L^2(\Omega)}$. Dividing by $\|u\|_{L^2(\Omega)}$, one obtains the desired inequality. ∎

We remark that, in general, $\|\mathcal{G}\|_{L^2(\Omega)\leftarrow L^2(\Omega)}$ cannot be replaced by the weaker Hilbert-Schmidt norm $\|\mathcal{G}\|_{\mathrm{F}} = \|G\|_{L^2(\Omega\times\Omega)}$, since G with its singularity (5.24a) is not square-integrable in $\Omega \times \Omega$ for $d \geq 4$.

5.4.4 Galerkin Discretisation of $\mathcal{G}$ and the Connection to A^{-1}

The Galerkin discretisation matrix of $\mathcal{G}$ from (5.25) is $B := R\mathcal{G}P$ with the components

$$B_{ij} := \int_\Omega \int_\Omega \varphi_i(x) G(x,y) \varphi_j(y) \mathrm{d}x\mathrm{d}y \qquad (i,j \in \mathcal{I}), \tag{5.27}$$

where we use the same finite-element basis φ_i as above.

Two different finite-element error estimates can be considered. The $L^2(\Omega)$ orthogonal projection is defined by

$$Q_h := PM^{-1}R : L^2(\Omega) \to V_h, \quad \text{i.e.,} \quad (Q_h u, v_h)_{L^2} = (u, v_h)_{L^2} \quad \text{for all } u \in V \text{ and } v_h \in V_h$$

(M is the mass matrix). The corresponding error is

$$e_h^Q(u) := \|u - Q_h u\|_{L^2(\Omega)}.$$

On the other hand, the finite-element approximation is associated with the *Ritz projection*

$$Q_{\mathrm{Ritz},h} = PA^{-1}RL : V \to V_h.$$

If $u \in V$ is the solution of the variational problem $a(u,v) = f(v)$ (see (5.2)), then $u_h = P_h u$ is the finite-element solution. The finite-element error is

$$e_h^P(u) := \|u - Q_{\mathrm{Ritz},h} u\|_{L^2(\Omega)}.$$

Since the $L^2(\Omega)$ orthogonal projection is optimal, i.e., $e_h^Q(u) \leq e_h^P(u)$, it suffices to bound the error e_h^P. The weakest form of finite-element convergence reads

$$e_h^P(u) \leq \varepsilon_h \|f\|_{L^2(\Omega)} \qquad \text{for all } u = \mathcal{G}f,\ f \in L^2(\Omega), \tag{5.28}$$

where $\varepsilon_h \to 0$ for $h \to 0$. This estimate will be provided by Lemma 5.11, whose presumptions hold for the problem considered here. Only under further smoothness conditions and regularity assumptions, one can expect a better behaviour like $\varepsilon_h = \mathcal{O}(h^\sigma)$ with $\sigma \in (0,2]$.

Lemma 5.11 *Let the bilinear form $a : V \times V \to \mathbb{R}$ be bounded, while the subspaces $V_n \subset V$ satisfy*

$$\lim_{n\to\infty} \operatorname{dist}(u, V_n) = 0 \qquad \textit{for all } u \in V \qquad (\operatorname{dist}(u,V_n) := \inf_{v\in V_n} \|u - v\|_V). \tag{5.29}$$

Let the discretisation $\{A_n\}_{n\in\mathbb{N}'}$ be stable and the embedding $V \hookrightarrow L^2(\Omega)$ be continuous, dense and compact. Then

$$\|L^{-1} - PA^{-1}R\|_{L^2(\Omega)\leftarrow L^2(\Omega)} \leq \varepsilon_n \qquad \textit{with } \varepsilon_n \to 0 \textit{ for } n \to \infty$$

(here $L : V \to V'$ is the operator associated with a : $a(u,v) = \langle Lu, v\rangle_{V'\times V}$. The last inequality can also be written as $\|(I - Q_{\mathrm{Ritz},n})u\|_{L^2(\Omega)} \leq \varepsilon_n \|f\|_{L^2(\Omega)}$ with $\mathcal{A}u = f$.

Proof: a) Because of the embedding property of $V \hookrightarrow L^2(\Omega)$, also the embedding $E : L^2(\Omega) \hookrightarrow V'$ is a continuous, dense and compact embedding (cf. Hackbusch [38, Lemmata 6.3.9 and 6.4.5b]).

b) Let $e_n(u) := (I - Q_{\mathrm{Ritz},n})\, u$ for $u \in V$. Cea's Lemma shows $\|e_n(u)\|_V \le C_1 \operatorname{dist}(u, V_n)$ (cf. Hackbusch [38, Theorem 8.2.1]). Hence, (5.29) and part a) imply $\|e_n(u)\|_{L^2(\Omega)} \to 0$ for all $u \in V$.

c) The stability of $\{A_n\}_{n\in\mathbb{N}'}$ together with (5.29) proves $L^{-1} \in \mathcal{L}(V', V)$ (cf. Hackbusch [38, Theorem 8.2.2]).

d) The set $U := \{L^{-1}f \in V : \|f\|_{L^2(\Omega)} \le 1\}$ is image of the unit sphere $\{f \in L^2(\Omega) : \|f\|_{L^2(\Omega)} \le 1\}$ under the mapping $L^{-1}E$. Since L^{-1} is bounded (see part c) and E is compact (see part a), U is a precompact subset of V.

e) Now we want to show the uniform convergence $\varepsilon_n := \sup\{\|e_n(u)\|_{L^2(\Omega)} : u \in U\} \to 0$. For the indirect proof assume that there are $\eta > 0$ and a sequence $u^{(n)} \in U$ such that

$$\|e_n(u^{(n)})\|_{L^2(\Omega)} \ge \eta > 0 \qquad \text{for all } n \in \mathbb{N}' \subset \mathbb{N}.$$

By the precompactness of U there is a subsequence $u^{(n)} \in U$ with $u^{(n)} \to u^* \in V$ for $n = n_k \to \infty$. Since $e_n(u^{(n)}) = e_n(u^{(n)} - u^*) + e_h^P(u^*)$, it follows that $\|e_n(u^{(n)})\|_{L^2(\Omega)} \le \|e_n(u^{(n)} - u^*)\|_{L^2(\Omega)} + \|e_n(u^*)\|_{L^2(\Omega)}$. For the first term use

$$\|e_n(u^{(n)} - u^*)\|_{L^2(\Omega)} \le C_0\|e_n(u^{(n)} - u^*)\|_V \le C_0C_1 \operatorname{dist}(u^{(n)} - u^*, V_n) \le C_0C_1\|u^{(n)} - u^*\|_V \to 0,$$

and $\|e_n(u^*)\|_{L^2(\Omega)} \le C_0\|e_n(u^*)\|_V \le C_0C_1 \operatorname{dist}(u^*, V_n) \to 0$ for the second. This yields the contradiction $\|e_n(u^{(n)})\|_{L^2(\Omega)} \to 0$. ∎

The next lemma shows that $M^{-1}BM^{-1}$ is an approximation of A^{-1}.

Lemma 5.12 *Let ε_h be as in (5.28). Then, using the norm $|||\cdot|||$ from (5.7), the following estimate holds:*

$$|||MA^{-1}M - B||| = \|PA^{-1}R - PM^{-1}BM^{-1}R\|_{L^2(\Omega)\leftarrow L^2(\Omega)} \le 2\varepsilon_h. \tag{5.30}$$

Proof: $|||MA^{-1}M - B||| = \|PA^{-1}R - PM^{-1}BM^{-1}R\|_{L^2(\Omega)\leftarrow L^2(\Omega)}$ follows from the definition (5.7). Because of $B = R\mathcal{G}P$, we have

$$PM^{-1}BM^{-1}R = PM^{-1}R\mathcal{G}PM^{-1}R = Q_h\mathcal{G}Q_h.$$

We have to estimate $\left(PA^{-1}R - Q_h\mathcal{G}Q_h\right) f$ for $f \in L^2(\Omega)$. Since $Rf = 0$ for $f \in V_h^\perp$, it suffices to use $f \in V_h$. $u_h := PA^{-1}Rf$ is the finite-element solution of $Lu = f$, while $Q_h\mathcal{G}Q_hf = Q_h\mathcal{G}f = Q_hu$ is the $L^2(\Omega)$ projection of the u into V_h. The expression can be formulated by means of the Ritz projection P_h,

$$\left(PA^{-1}R - Q_h\mathcal{G}Q_h\right) f = u_h - Q_hu = Q_{\mathrm{Ritz},h}u - Q_hu = (u - Q_hu) - (u - Q_{\mathrm{Ritz},h}u)$$

and can be bounded by $\|\left(PA^{-1}R - Q_h\mathcal{G}Q_h\right) f\|_{L^2(\Omega)} \le e_h^Q(u) + e_h^P(u) \le 2e_h^P(u) \le 2\varepsilon_h\|f\|_{L^2(\Omega)}$. This proves the inequality (5.30). ∎

Corollary 5.13 *An equivalent formulation of the triple norm in (5.30) is*

$$|||MA^{-1}M - B||| = \|M^{1/2}A^{-1}M^{1/2} - M^{-1/2}BM^{-1/2}\|_2.$$

This implies $\|A^{-1} - M^{-1}BM^{-1}\|_2 \le |||MA^{-1}M - B|||\, \|M^{-1}\|_2 \le 2\|M^{-1}\|_2\varepsilon_h$.

We remark that the boundedness of $\mathcal{G}$ implies the boundedness of B. Because of

$$|||B||| = \|Q_h\mathcal{G}Q_h\|_{L^2(\Omega)\leftarrow L^2(\Omega)}$$

we conclude the inequality in

Remark 5.14 $|||B||| \le \|\mathcal{G}\|_{L^2(\Omega)\leftarrow L^2(\Omega)}$.

The further steps are as follows. In §5.5 we show that B can be approximated by an $\mathcal{H}$-matrix $B_\mathcal{H}$. Theorem 5.8 states the M^{-1} has an $\mathcal{H}$-matrix approximation $N_\mathcal{H}$. By Theorem 7.17 the product $N_\mathcal{H}B_\mathcal{H}N_\mathcal{H}$ is again an $\mathcal{H}$-matrix which now approximates $M^{-1}BM^{-1}$. Since additional errors of the size of the discretisation error ε_h are acceptable, approximations of $M^{-1}BM^{-1}$ are also good approximations of A^{-1} (see (5.30)).

5.4.5 Conclusions from the Separable Approximation of Green's Function

We anticipate the result of Section §5.5 and assume that the Green function allows a separable approximation

$$G(x,y) \approx G_k(x,y) = \sum_{i=1}^{k} u_i^{(k)}(x) v_i^{(k)}(y) \qquad \text{for } x \in X, \quad y \in Y, \tag{5.31a}$$

where the subsets $X, Y \subset \Omega$ satisfy the usual admissibility condition

$$\min\{\operatorname{diam}(X), \operatorname{diam}(Y)\} \le \eta \operatorname{dist}(X,Y). \tag{5.31b}$$

Moreover, we assume that the approximation error decays exponentially, i.e., the integral operators $\mathcal{G}_{XY}, \mathcal{G}_{k,XY} \in \mathcal{L}\left(L^2(Y), L^2(X)\right)$ defined by

$$\left(\mathcal{G}_{XY} u\right)(x) = \int_Y G(x,y) u(y) \mathrm{d}y, \qquad \left(\mathcal{G}_{k,XY} u\right)(x) = \int_Y G_k(x,y) u(y) \mathrm{d}y \qquad (x \in X)$$

satisfy

$$\|\mathcal{G}_{XY} - \mathcal{G}_{k,XY}\|_{L^2(X) \leftarrow L^2(Y)} \le \varepsilon \|\mathcal{G}\|_{L^2(\Omega) \leftarrow L^2(\Omega)} \quad \text{with } \varepsilon = \varepsilon(k) \le C_1 \exp(-c_2 k^{c_3}),\ c_1, c_2, c_3 > 0 \tag{5.31c}$$

for all $k \in \mathbb{N}$. We shall prove (5.31c) with the constants

$$c_1 \approx 1, \quad c_2 \approx c_\eta^{d/(d+1)}, \quad c_3 = \frac{1}{d+1} \qquad (c_\eta = \beta_0 \mathrm{e} \text{ with } \beta_0 \text{ from (5.47d) below}). \tag{5.31d}$$

The proof, however, uses a further assumption: If the minimum in (5.31b) is taken by $\operatorname{diam}(X)$ [$\operatorname{diam}(Y)$], then X [Y] must be convex. In practice this is no restriction, since by construction of the partition $P \subset T(I \times I)$ the admissibility of the bounding boxes are used and the bounding boxes are convex. The verification of (5.31a-d) follows after the proof of Lemma 5.28.

The matrix B from (5.27) is obviously approximated by $B_k \in \mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, k)$, where the definition of $B_k|_b$ uses the approximation G_k from (5.31a). For $b = (t,s)$, the sets X and Y in (5.31a) become Ω_t and Ω_s. The estimate (5.31c) implies that

$$\|\mathcal{G}_{\Omega_t,\Omega_s} - \mathcal{G}_{k,\Omega_t,\Omega_s}\|_{L^2(\Omega_t) \leftarrow L^2(\Omega_s)} \le \epsilon \|\mathcal{G}\|_{L^2(\Omega) \leftarrow L^2(\Omega)} \qquad \text{holds for all } b = (t,s) \in \mathcal{L}^+(T_{\mathcal{I}\times\mathcal{I}})$$

while no approximation is used in inadmissible leaves $b = (t,s) \in \mathcal{L}^-(T_{\mathcal{I}\times\mathcal{I}})$. The total error can be estimated by means of Theorem 5.4: for a sparse block cluster tree and an L^2-stable finite element basis, we get

$$|||B - B_k||| \le \varepsilon C_{\mathrm{fe}} C_{\mathrm{sp}} (1 + \operatorname{depth}(T_{\mathcal{I}\times\mathcal{I}})) \|\mathcal{G}\|_{L^2(\Omega) \leftarrow L^2(\Omega)}$$

This inequality combined with Lemma 7.26, which yields $C_{\mathrm{sp}} = \mathcal{O}(1)$, and Lemma 5.26, which yields $\|\mathcal{G}\|_{L^2(\Omega) \leftarrow L^2(\Omega)} = \mathcal{O}(1)$, show

$$|||B - B_k||| \le \mathcal{O}(\varepsilon \operatorname{depth}(T_{\mathcal{I}})).$$

This proves the following Lemma:

Lemma 5.15 *The matrix $B_k \in \mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, k)$ defined above satisfies the error estimate*

$$|||B - B_k||| \le \mathcal{O}(\varepsilon \cdot \operatorname{depth}(T_{\mathcal{I}})) \qquad \text{with } \varepsilon = \varepsilon(k) \le \exp(-c_2 k^{1/(d+1)})$$

for all $k \in \mathbb{N}$ and with the constant c_2 from (5.31d).

Remark 5.16 *a) The usual construction of the cluster tree leads to* $\operatorname{depth}(T_{\mathcal{I}}) = \mathcal{O}(\log \#\mathcal{I})$.

b) In the following we choose $k_{\varepsilon,B} \ge \mathcal{O}\left(\log^{d+1}((1 + \operatorname{depth}(T_{\mathcal{I}}))/\varepsilon)\right)$, so that

$$|||B - B_k||| \le \varepsilon \qquad \text{for } k = k_{\varepsilon,B}. \tag{5.32}$$

c) $|||B||| \le \|\mathcal{G}\|_{L^2(\Omega) \leftarrow L^2(\Omega)} = \mathcal{O}(1)$ from Remark 5.14 and Lemma 5.10 show that $\varepsilon \le \mathcal{O}(1)$ leads to

$$|||B_k||| \le \mathcal{O}(1). \tag{5.33}$$

Since $A^{-1} \approx M^{-1}BM^{-1}$ (cf. Lemma 5.12) and $M^{-1} \approx N_{\mathcal{H}}$ (cf. Theorem 5.8) as well as $B \approx B_k$, we use

$$H := N_{\mathcal{H}} B_{k_{\varepsilon,B}} N_{\mathcal{H}} \tag{5.34}$$

as approximation of the inverse finite-element matrix A^{-1}.

Lemma 5.17 *Given $\varepsilon > 0$, choose $N_{\mathcal{H}} \in \mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, k_\varepsilon)$ according to Theorem 5.8 and $B_{k_{\varepsilon,B}} \in \mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, k_{\varepsilon,B})$ according to Lemma 5.15. According to Theorem 7.17, the (exact) product H from (5.34) is a hierarchical matrix from $\mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, k_{\varepsilon,H})$ with*

$$k_{\varepsilon,H} = C_{\mathrm{U}}^2 \max\{k_\varepsilon, k_{\varepsilon,B}, n_{\min}\}, \tag{5.35}$$

for $C_{\mathrm{U}} = \mathcal{O}(\log \#\mathcal{I})$.

Proof: According to Theorem 5.8, the exact product $N_{\mathcal{H}} B_{k_{\varepsilon,B}}$ satisfies

$$N_{\mathcal{H}} B_{k_{\varepsilon,B}} \in \mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, k_{NB}) \qquad \text{with } k_{NB} = C_{\mathrm{U}} \max\{k_\varepsilon, k_{\varepsilon,B}, n_{\min}\}.$$

The second product $H = \left(N_{\mathcal{H}} B_{k_{\varepsilon,B}}\right) N_{\mathcal{H}}$ leads to $k_{\varepsilon,H} = C_{\mathrm{U}} \max\{k_{NB}, k_\varepsilon, n_{\min}\}$. Since $C_{\mathrm{U}} \geq 1$, one concludes from $\max\{k_\varepsilon, n_{\min}\} \leq k_{NB}$ the statement (5.35). ∎

The appropriate norm for the error $A^{-1} - H$ is

$$\|P\left(A^{-1} - H\right) R\|_{L^2(\Omega)\leftarrow L^2(\Omega)} = \|M^{1/2}\left(A^{-1} - H\right) M^{1/2}\|_2 = |||MA^{-1}M - B|||. \tag{5.36a}$$

(cf. (5.7)). Multiple application of the triangle inequality yields

$$\begin{aligned}\|P\left(A^{-1} - H\right) R\|_{L^2(\Omega)\leftarrow L^2(\Omega)} \leq{}& \|P\left(A^{-1} - M^{-1}BM^{-1}\right) R\|_{L^2(\Omega)\leftarrow L^2(\Omega)} \\ &+ \|PM^{-1}\left(B - B_{k_{\varepsilon,B}}\right) M^{-1}R\|_{L^2(\Omega)\leftarrow L^2(\Omega)} \\ &+ \|PM^{-1}B_{k_{\varepsilon,B}}\left(M^{-1} - N_{\mathcal{H}}\right) R\|_{L^2(\Omega)\leftarrow L^2(\Omega)} \\ &+ \|P\left(M^{-1} - N_{\mathcal{H}}\right) B_{k_{\varepsilon,B}} N_{\mathcal{H}} R\|_{L^2(\Omega)\leftarrow L^2(\Omega)}.\end{aligned} \tag{5.36b}$$

The first term in (5.36b) is estimated in (5.30) by means of the quantity ε_h defined there:

$$\|P\left(A^{-1} - M^{-1}BM^{-1}\right) R\|_{L^2(\Omega)\leftarrow L^2(\Omega)} \leq 2\varepsilon_h. \tag{5.36c}$$

The second term $\|PM^{-1}\left(B - B_{k_{\varepsilon,B}}\right) M^{-1}R\|_{L^2(\Omega)\leftarrow L^2(\Omega)} = |||B - B_{k_{\varepsilon,B}}|||$ in (5.36b) can be treated by Lemma 5.15 and (5.32):

$$\|PM^{-1}\left(B - B_{k_{\varepsilon,B}}\right) M^{-1}R\|_{L^2(\Omega)\leftarrow L^2(\Omega)} \leq \varepsilon. \tag{5.36d}$$

The third term in (5.36b) is split into the factors

$$\begin{aligned}&\|PM^{-1}B_{k_{\varepsilon,B}}\left(M^{-1} - N_{\mathcal{H}}\right) R\|_{L^2(\Omega)\leftarrow L^2(\Omega)} \\ &= \|\left[PM^{-1}B_{k_{\varepsilon,B}}M^{-1/2}\right]\left[M^{1/2}\left(M^{-1} - N_{\mathcal{H}}\right) R\right]\|_{L^2(\Omega)\leftarrow L^2(\Omega)} \\ &\leq \|PM^{-1}B_{k_{\varepsilon,B}}M^{-1/2}\|_{L^2(\Omega)\leftarrow\mathbb{R}^{\mathcal{I}}}\|M^{1/2}\left(M^{-1} - N_{\mathcal{H}}\right) R\|_{\mathbb{R}^{\mathcal{I}}\leftarrow L^2(\Omega)} \\ &= |||B_{k_{\varepsilon,B}}|||\|M^{1/2}(M^{-1} - N_{\mathcal{H}})M^{1/2}\|_2 \\ &\leq \mathcal{O}(1)\cdot\varepsilon = \mathcal{O}(\varepsilon),\end{aligned} \tag{5.36e}$$

where the last inequality follows from (5.33) and (5.18c).

The fourth term in (5.36b) is treated analogously:

$$\begin{aligned}&\|P(M^{-1} - N_{\mathcal{H}})B_{k_{\varepsilon,B}}N_{\mathcal{H}}R\|_{L^2(\Omega)\leftarrow L^2(\Omega)} \\ &= \left\|\left[P\left(M^{-1} - N_{\mathcal{H}}\right) M^{1/2}\right]\left[M^{-1/2}B_{k_{\varepsilon,B}}N_{\mathcal{H}}R\right]\right\|_{L^2(\Omega)\leftarrow L^2(\Omega)} \\ &\leq \|P\left(M^{-1} - N_{\mathcal{H}}\right) M^{1/2}\|_{L^2(\Omega)\leftarrow\mathbb{R}^{\mathcal{I}}}\|M^{-1/2}B_{k_{\varepsilon,B}}N_{\mathcal{H}}R\|_{\mathbb{R}^{\mathcal{I}}\leftarrow L^2(\Omega)}\end{aligned}$$

$$
\begin{aligned}
&= \|M^{1/2}\left(M^{-1} - N_{\mathcal{H}}\right) M^{1/2}\|_2 \|PM^{-1}B_{k_{\varepsilon,B}} N_{\mathcal{H}} R\|_{L^2(\Omega)\leftarrow L^2(\Omega)} \\
&= \varepsilon \|PM^{-1}B_{k_{\varepsilon,B}} N_{\mathcal{H}} R\|_{L^2(\Omega)\leftarrow L^2(\Omega)} \\
&\le \varepsilon \left(\|PM^{-1}B_{k_{\varepsilon,B}} M^{-1} R\|_{L^2(\Omega)\leftarrow L^2(\Omega)} + \|PM^{-1}B_{k_{\varepsilon,B}} (M^{-1} - N_{\mathcal{H}}) R\|_{L^2(\Omega)\leftarrow L^2(\Omega)} \right).
\end{aligned}
$$

The first norm expression in the last line is $\|PM^{-1}B_{k_{\varepsilon,B}} M^{-1} R\|_{L^2(\Omega)\leftarrow L^2(\Omega)} = |||B_k||| \le \mathcal{O}(1)$, the second is estimated in (5.36e) by $\mathcal{O}(\varepsilon)$, so that

$$
\|P\left(M^{-1} - N_{\mathcal{H}}\right) B_{k_{\varepsilon,B}} N_{\mathcal{H}} R\|_{L^2(\Omega)\leftarrow L^2(\Omega)} \le \mathcal{O}(\varepsilon). \tag{5.36f}
$$

The combination of (5.36a-f) yields the final result.

Theorem 5.18 *Let the assumption of the previous Sections §5.2, §5.3, §5.4.1 hold and assume* $\operatorname{depth}(T_{\mathcal{I}}) = \mathcal{O}(\#I)$. *Let* $\varepsilon \in (0,1)$ *be given. Choosing the local rank* $k_\varepsilon = \mathcal{O}(\log^d(\#\mathcal{I}/\varepsilon))$ *of* $N_{\mathcal{H}} \in \mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, k_\varepsilon)$ *and* $k_{\varepsilon,B} = \mathcal{O}(\log^{d+1}(\#\mathcal{I}/\varepsilon))$ *of* $B_{k_{\varepsilon,B}} \in \mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, k_{\varepsilon,B})$, *one obtains an approximation* $H = N_{\mathcal{H}} B_{k_{\varepsilon,B}} N_{\mathcal{H}} \in \mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, k_{\varepsilon,H})$ *of the inverse finite-element matrix* A^{-1} *with the error*

$$
\|P(A^{-1} - H)R\|_{L^2(\Omega)\leftarrow L^2(\Omega)} \le \mathcal{O}(\varepsilon + \varepsilon_h),
$$

where $k_{\varepsilon,H} = C_{\mathrm{U}}^2 \max\{k_\varepsilon, k_{\varepsilon,B}, n_{\min}\} = \mathcal{O}\left(\log^2(\#\mathcal{I}) k_{\varepsilon,B}\right) = \mathcal{O}\left(\log^{d+3}(\#\mathcal{I}) + \log^2(\#\mathcal{I}) \log^{d+1}(1/\varepsilon)\right)$, *while* ε_h *is the finite-element consistency error from (5.28).*

An obvious choice of ε is $\varepsilon = \varepsilon_h$. Since, in the best case, $\varepsilon_h = \mathcal{O}(h^\alpha) = \mathcal{O}(\#\mathcal{I}^{-\alpha/d})$ ($\alpha > 0$: consistency order) the quantities $\log(\#\mathcal{I})$ and $\log(1/\varepsilon)$ coincide with respect to their size and yield

$$
\|P\left(A^{-1} - H\right) R\|_{L^2(\Omega)\leftarrow L^2(\Omega)} \le \mathcal{O}(h^\alpha)
$$

for an $H \in \mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, k_{\varepsilon,H})$ with $k_{\varepsilon,H} = \mathcal{O}\left(\log^{d+3}(\#\mathcal{I})\right)$.

5.5 Analysis of the Green Function

The goal of this Section is to construct a separable approximation $G(x,y) \approx G^{(k)}(x,y) = \sum_{i=1}^k u_i(x) v_i(y)$ of Green's function in $X \times Y$ (see (5.31a)), where $X, Y \subset \Omega$ satisfy the admissibility condition.

The Green function G is defined in $\Omega \times \Omega$. If $X \subset \Omega$ and $Y \subset \Omega$ are disjoint, the restriction of G to $X \times Y$ is L-harmonic, i.e., $LG = 0$. The subspace of L-harmonic functions will be considered in §5.5.2. First we give approximation results for general closed subspaces of $L^2(D)$.

5.5.1 Approximation by Finite Dimensional Subspaces

In the following $D \subset \mathbb{R}^d$ is a domain. All distances and diameters use the Euclidean norm in $\mathbb{R}^d$ except the distance of functions which uses the $L^2(D)$-norm.

Lemma 5.19 *Let* $D \subset \mathbb{R}^d$ *be a convex domain and* Z *a closed subspace of* $L^2(D)$. *Then for any* $k \in \mathbb{N}$ *there is a subspace* $V_k \subset Z$ *satisfying* $\dim V_k \le k$ *so that*

$$
\operatorname{dist}_{L^2(D)}(u, V_k) \le c_{\mathrm{appr}} \frac{\operatorname{diam}(D)}{\sqrt[d]{k}} \|\nabla u\|_{L^2(D)} \qquad \text{for all } u \in Z \cap H^1(D), \text{ where } c_{\mathrm{appr}} := \frac{2\sqrt{d}}{\pi}. \tag{5.37}
$$

Proof: a) D is contained in a cube with side length $\operatorname{diam}(D)$. Let z be the centre point:

$$
D \subset Q = \{x \in \mathbb{R}^d : \|x - z\|_\infty < \frac{1}{2} \operatorname{diam}(D)\}.
$$

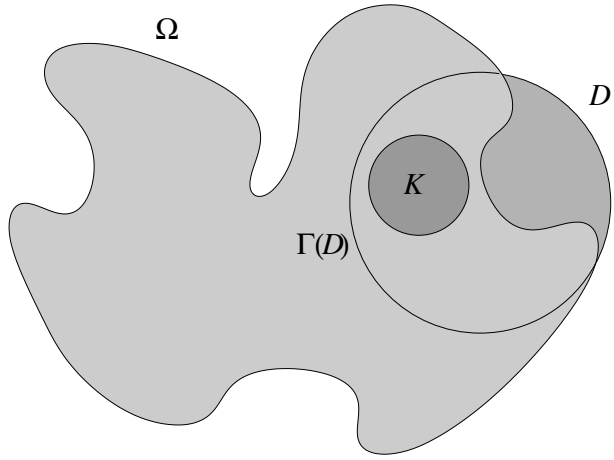


Figure 5.1: Domains Ω, D and $\Gamma(D)$

b) First let $k = \ell^d$. We subdivide the cube Q uniformly into k subcubes Q_i of side length $\operatorname{diam}(D)/\ell$ and set $D_i = D \cap Q_i$ $(i = 1, \dots, k)$. Each D_i is convex with $\operatorname{diam}(D_i) \le \frac{\sqrt{d}}{\ell} \operatorname{diam}(D)$. The subspace

$$W_k = \{v \in L^2(D) : v \text{ is constant on } D_i \text{ for all } i = 1, \dots, k\}$$

has the dimension $\dim W_k \le k$. Poincaré's inequality for $u \in H^1(D)$ shows that

$$\int_{D_i} |u - \bar{u}_i|^2 dx \le \Big(\frac{\operatorname{diam}(D_i)}{\pi}\Big)^2 \int_{D_i} |\nabla u|^2 dx \le \Big(\frac{\sqrt{d}\operatorname{diam}(D)}{\pi\ell}\Big)^2 \int_{D_i} |\nabla u|^2 dx,$$

where $\bar{u}_i = \operatorname{vol}(D_i)^{-1} \int_{D_i} u\, dx$ is the mean value of u in D_i. Summation over all i yields

$$\operatorname{dist}_{L^2(D)}(u, W_k) \le \|u - \bar{u}\|_{L^2(D)} \le \frac{\sqrt{d}}{\pi\ell} \operatorname{diam}(D)\, \|\nabla u\|_{L^2(D)},$$

where $\bar{u}$ is the piecewise constant function from W_k with $\bar{u}|_{D_i} = \bar{u}_i$.

c) For general $k \in \mathbb{N}$, choose $\ell := \lfloor \sqrt[d]{k} \rfloor \in \mathbb{N}$, i.e., $\ell^d \le k < (\ell + 1)^d$. Applying Part (a) for $k' := \ell^d$, we use the space $W_k := W_{k'}$ satisfying $\dim W_k = \dim W_{k'} \le k' \le k$. Using $\frac{1}{\ell} \le \frac{2}{\ell+1} < \frac{2}{\sqrt[d]{k}}$, we arrive at

$$\operatorname{dist}_{L^2(D)}(u, W_k) \le c_{\text{appr}} \frac{\operatorname{diam}(D)}{\sqrt[d]{k}} \|\nabla u\|_{L^2(D)}$$

with the constant $c_{\text{appr}} := 2\sqrt{d}/\pi$.

d) Let $\Pi : L^2(D) \to X$ be the $L^2(D)$-orthogonal projection onto X and $V_k = \Pi(W_k)$. Keeping in mind that P has norm one and $u \in X$, the assertion follows from $\|u - \Pi\bar{u}\|_{L^2(D)} = \|\Pi(u - \bar{u})\|_{L^2(D)} \le \|u - \bar{u}\|_{L^2(D)}$ for all $\bar{u} \in W_k$. ∎

In the last proof we have restricted D_i to convex domains though Poincaré's inequality holds whenever the embedding $H^1(D_i) \hookrightarrow L^2(D_i)$ is compact. This is for example true if D_i fulfils a uniform cone condition. However, in the general case extra assumptions are required to ensure the uniform boundedness of the Poincaré constants for all D_i.

5.5.2 Space $Z(D)$ of L-Harmonic Functions

$\Omega \subset \mathbb{R}^d$ is the domain where the differential operator L is defined (cf. (5.19), (5.20)). Let $D \subset \mathbb{R}^d$ be a further domain with $\Omega \cap D \neq \emptyset$. In the following we will specify the space $Z = Z(D)$ which appears in

Lemma 5.19. $Z(D)$ has to satisfy three properties: $u \in Z(D)$ should be (i) L-harmonic, (ii) locally in H^1, and if (iii) $D \not\subset \Omega$, $u = 0$ holds in $D\backslash\Omega$. Before we give the precise definition, we add some remarks:

ad (i): A function satisfying $Lu = 0$ in a subdomain, is called there *L-harmonic*. In particular, the Green function $G(x, y)$ is L-harmonic with respect of both arguments x, y, if $x \neq y$. The appropriate description (5.39c) uses the weak formulation by means of the bilinear form a from (5.2).

ad (ii): If $x \in \Omega \cap \partial D$, the function $u := G(x, \cdot)$ is L-harmonic in D (since $x \notin D$), but due to the singularity of the Green function u does not belong to $H^1(D)$. There we require only a *local* H^1-property: $u \in H^1(K)$ for all domains $K \subset D$, which have a positive distance from the interior boundary

$$\Gamma(D) := \Omega \cap \partial D \tag{5.38}$$

(see Figure 5.1 and (5.39b)).

ad (iii): For later constructions it is helpful if D is not restricted to subsets of Ω. Because of the homogeneous boundary condition $G(x, y) = 0$ for $x \in \Omega$ and $y \in \partial\Omega$, the continuation of $G(x, \cdot)$ by zero in $D\backslash\Omega$ belongs again locally to H^1. Condition (5.39a) is an empty statement if $D \subset \Omega$ holds.

Definition 5.20 *Let $\Omega, D \subset \mathbb{R}^d$ and $\Gamma(D)$ be defined as above. Let $Z(D)$ be the space of all $u \in L^2(D)$ with the properties*

$$u|_{D\backslash\Omega} = 0, \tag{5.39a}$$

$$u \in H^1(K) \qquad \text{for all } K \subset D \text{ with } \operatorname{dist}(K, \Gamma(D)) > 0, \tag{5.39b}$$

$$a(u, \varphi) = 0 \qquad \text{for all } \varphi \in H^1_0(D \cap \Omega). \tag{5.39c}$$

Lemma 5.21 *The subspace $Z(D)$ is closed in $L^2(D)$.*

This statement is needed to apply Lemma 5.19 to $Z = Z(D)$. Its proof is postponed to the subsection, following Lemma 5.23.

Remark 5.22 *Let $Z(D)$ and $Z(D')$ be subspaces for $D' \subset D$. For all $u \in Z(D)$ the restriction $u|_{D'}$ belongs to $Z(D')$. The short notation of this statement is $Z(D)|_{D'} \subset Z(D')$. If $\operatorname{dist}(D', \Gamma(D)) > 0$, even $Z(D)|_{D'} \subset Z(D') \cap H^1(D')$ holds (cf. (5.39b)).*

5.5.3 Inner Regularity

Inner regularity denotes the characteristic property of homogeneous solutions of elliptic partial differential equations, to have better regularity in inner subdomains. Here, we make use of the fact that for functions $u \in Z(D)$ the gradient norm $\|\nabla u\|_{L^2(K\cap\Omega)}$ in a smaller domain K can be estimated by means of $\|u\|_{L^2(D\cap\Omega)}$. Because of the continuation by zero in $D\backslash\Omega$ (cf. (5.39a)) also the norms $\|\nabla u\|_{L^2(K)}$ and $\|u\|_{L^2(D)}$ can be used.

Lemma 5.23 *Let Ω, D, $Z(D)$, $\Gamma(D)$, and $K \subset D$ with $\operatorname{dist}(K, \Gamma(D)) > 0$ as in Definition 5.20. $\kappa_C = \lambda_{\max}/\lambda_{\min}$ is the quantity from (5.21)). Then the so-called* Caccioppoli inequality *holds:*

$$\|\nabla u\|_{L^2(K\cap\Omega)} \leq \frac{2\sqrt{\kappa_C}}{\operatorname{dist}(K, \Gamma(D))} \|u\|_{L^2(D\cap\Omega)} \qquad \text{for all } u \in Z(D). \tag{5.40}$$

Proof: Let the cut-off function $\eta \in C^1(D)$ satisfy $0 \leq \eta \leq 1$, $\eta = 1$ in K, $\eta = 0$ in a neighbourhood of $\Gamma(D)$, and $|\nabla\eta| \leq 2/\delta$ in $D \cap \Omega$, where we use the abbreviation

$$\delta := \operatorname{dist}(K, \Gamma(D)).$$

Since $K' := \mathrm{supp}(\eta) \subset D$ satisfies the condition $\mathrm{dist}(K', \Gamma(D)) > 0$, one obtains from (5.39b) that $u \in H^1(K')$. Hence, $\varphi := \eta^2 u \in H_0^1(D \cap \Omega)$ is a possible test function in $a(u, \varphi) = 0$:

$$0 = \int_{D\cap\Omega} (\nabla u)^\top C(x) \nabla(\eta^2 u) dx = 2 \int_{D\cap\Omega} \eta u (\nabla u)^\top C(x)(\nabla\eta) dx + \int_{D\cap\Omega} \eta^2 (\nabla u)^\top C(x)(\nabla u) dx. \tag{5.41}$$

The inequality chain

$$\begin{aligned}
\int_{D\cap\Omega} \eta^2 \|C^{1/2}(x)\nabla u\|^2 \mathrm{d}x = \int_{D\cap\Omega} \eta^2 (\nabla u)^\top C(x)(\nabla u) \mathrm{d}x &\underset{(5.41)}{=} 2 \left| \int_{D\cap\Omega} \eta u (\nabla u)^\top C(x)(\nabla\eta) \mathrm{d}x \right| \\
&\leq 2 \int_{D\cap\Omega} \eta\, |u|\, \|C^{1/2}(x)\nabla\eta\|\, \|C^{1/2}(x)\nabla u\|\, \mathrm{d}x \\
&\underset{\substack{\|C\|\leq\lambda_{\max} \text{ because of (5.21), } |\nabla\eta|\leq 2/\delta}}{\leq} 4\, \frac{\sqrt{\lambda_{\max}}}{\delta} \int_{D\cap\Omega} |u|\, \eta\, \|C^{1/2}(x)\nabla u\|\, \mathrm{d}x \\
&\underset{\text{Schwarz inequality}}{\leq} 4\, \frac{\sqrt{\lambda_{\max}}}{\delta} \sqrt{\int_{D\cap\Omega} \eta^2 \|C^{1/2}(x)\nabla u\|^2 \mathrm{d}x}\, \|u\|_{L^2(D\cap\Omega)}
\end{aligned}$$

can be divided by $\sqrt{\int_{D\cap\Omega} \eta^2 \|C^{1/2}(x)\nabla u\|^2 \mathrm{d}x} = \|\eta\, C^{1/2}(x)\nabla u\|_{L^2(D\cap\Omega)}$:

$$\|\eta\, C^{1/2}(x)\nabla u\|_{L^2(D\cap\Omega)} \leq 4 \frac{\sqrt{\lambda_{\max}}}{\delta} \|u\|_{L^2(D\cap\Omega)}.$$

Since $\eta = 1$ in K, one concludes that

$$\|\nabla u\|_{L^2(K\cap\Omega)} = \|\eta\nabla u\|_{L^2(K\cap\Omega)} \leq \|\eta\nabla u\|_{L^2(D\cap\Omega)} \underset{(5.21)}{\leq} \lambda_{\min}^{-1/2} \|\eta C^{1/2}(x)\nabla u\|_{L^2(D\cap\Omega)}.$$

Altogether, the assertion (5.40) follows with the factor 4 instead of 2. The condition $|\nabla\eta| \leq 2/\delta$ can be replaced by $|\nabla\eta| \leq (1+\varepsilon)/\delta$ for any $\varepsilon > 0$. Therefore, (5.40) holds with the factor $2(1+\varepsilon)$ for any $\varepsilon > 0$, hence also for 2. ■

Proof: [Proof of Lemma 5.21] Let $\{u_k\}_{k\in\mathbb{N}} \subset Z(D)$ be a sequence converging in $L^2(D)$ to u. Let $K \subset D$ with $\mathrm{dist}(K, \Gamma(D)) > 0$. Since $\|u_k\|_{L^2(D)}$ is uniformly bounded, due to Lemma 5.23, also $\{\nabla u_k\}_{k\in\mathbb{N}}$ is uniformly bounded on K:

$$\|\nabla u_k\|_{L^2(K)} \leq c\, \|u_k\|_{L^2(D)} \leq C.$$

By the Theorem of Banach-Alaoglu, a subsequence $\{u_{i_k}\}_{k\in\mathbb{N}}$ converges weakly in $H^1(K)$ to $\hat{u} \in H^1(K)$. Hence, for each $v \in L^2(K)$ we have $(u, v)_{L^2(K)} = \lim_{k\to\infty} (u_{i_k}, v)_{L^2(K)} = (\hat{u}, v)_{L^2(K)}$ showing $u = \hat{u} \in H^1(K)$. Since for any $\varphi \in H_0^1(D\cap\Omega)$ the functional $a(\cdot, \varphi)$ belongs to $(H^1(K))'$, the same argument yields $a(u, \varphi) = 0$. Finally, $u_k|_{D\setminus\Omega} = 0$ implies also $u|_{D\setminus\Omega} = 0$, proving $u \in Z(D)$. ■

5.5.4 Main Theorem

In the following construction we start from a convex domain K with $K \cap \Omega \neq \emptyset$. Broadening of K by $r > 0$ yields

$$K(r) := \{x \in \mathbb{R}^d : \mathrm{dist}(x, K) < r\} \qquad \text{for } 0 < r \leq \delta, \tag{5.42}$$

where $K(0) := K$ is set for $r = 0$. Let D be a superset of K with

$$\delta := \mathrm{dist}(K, \Gamma(D)) > 0. \tag{5.43}$$

It is easy to see that $K(\delta) \cap \Omega \subset D \cap \Omega$. In the complement $\mathbb{R}^d \setminus \Omega$, where all functions are defined by zero, D can be enlarged in such a way that $K(\delta) \subset D$.

The following lemma describes the approximability of all $u \in Z(D)$ by means of a subspace $W \subset Z(K)$, so that the approximation error decays exponentially with respect to the dimension $\dim W$ (i.e., the dimension grows only logarithmically with the inverse approximation error).

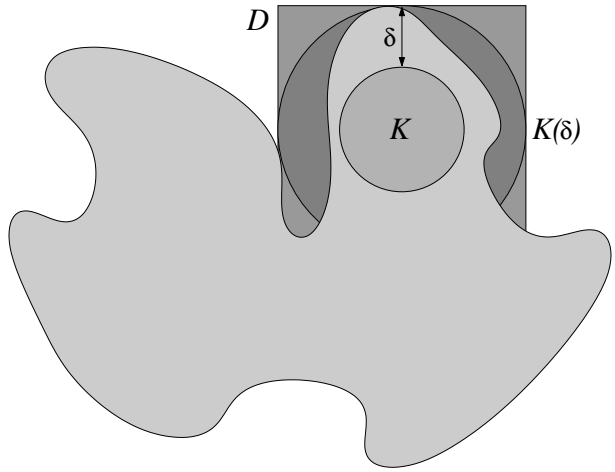


Figure 5.2: The domains Ω, $K \subset K(\delta) \subset D$ and $\Gamma(D)$

Lemma 5.24 *Let Ω, D, $Z(D)$, $\Gamma(D)$, and $K \subset D$ with $\operatorname{dist}(K, \Gamma(D)) > 0$ as in Definition 5.20. Furthermore, let K be a convex domain with*

$$\operatorname{diam}(K) \le \eta \operatorname{dist}(K, \Gamma(D)).$$

Then for all $\varepsilon < 1$ there exists a subspace $W = W_\varepsilon \subset Z(K)$ with the approximation property

$$\operatorname{dist}_{L^2(K)}(u, W) \le \varepsilon \|u\|_{L^2(D\cap\Omega)} \qquad \text{for all } u \in Z(D) \tag{5.44}$$

and the dimension

$$\dim W \le c_\eta^d \lceil \log \tfrac{1}{\varepsilon} \rceil^{d+1} + \lceil \log \tfrac{1}{\varepsilon} \rceil \qquad \text{with } c_\eta = 2\,\mathrm{e}\, c_{\mathrm{appr}} \sqrt{\kappa_C}\ (\eta + 2)\,. \tag{5.45}$$

Proof: a) Convexity of K implies also convexity of all $K(r)$ from (5.42), which for increasing r become larger: $K(r_1) \supset K(r_2)$ for $r_1 \ge r_2$. The smallest domain is $K(0) = K$, while $K(\delta)$ is the maximal domain with $K(\delta) \subset D$. It is easy to see that

$$\operatorname{dist}(K(r_2), \partial K(r_1)) = r_1 - r_2 \quad \text{for } r_1 \ge r_2 \qquad \text{and} \quad \operatorname{diam}(K(r)) \le \operatorname{diam}(K) + 2r. \tag{5.46a}$$

b) We fix a $p \in \mathbb{N}$, which will be specified in part f), and introduce radii $r_0 > r_1 > \ldots > r_p = 0$ by means of

$$r_j := (1 - \frac{j}{p})\delta \qquad (0 \le j \le p)\,. \tag{5.46b}$$

We define

$$K_j := K(r_j), \quad Z_j := Z(K_j) \qquad \text{(compare Definition 5.20)}$$

and remark that $K = K_p \subset K_{p-1} \subset \ldots \subset K_1 \subset K_0 \subset D$.

c) Let $j \in \{1, \ldots, p\}$. Application of Lemma 5.23 to the domains K_{j-1}, K_j instead of D, K yields

$$\|\nabla v\|_{L^2(K_j)} \le \frac{2\sqrt{\kappa_C}}{\operatorname{dist}(K_j, \Gamma(K_{j-1}))} \|v\|_{L^2(K_{j-1})} \qquad \text{for all } v \in Z_{j-1},$$

where the notation $\Gamma(K_{j-1})$ is explained in (5.38). Because of $\operatorname{dist}(K_j, \Gamma(K_{j-1})) \ge \operatorname{dist}(K_j, \partial K_{j-1}) = r_{j-1} - r_j = \delta/p$ (see (5.46a)) it follows that

$$\|\nabla v\|_{L^2(K_j)} \le \frac{2p\sqrt{\kappa_C}}{\delta} \|v\|_{L^2(K_{j-1})} \qquad \text{for all } v \in Z_{j-1}. \tag{5.47a}$$

d) We apply Lemma 5.19 with K_j instead of D and with $k := \lceil (\beta p)^d \rceil$. The factor β will be specified later in (5.47d). Following Lemma 5.19, there is a subspace $V_j \subset Z_j$ of dimension $\dim V_j \le k$ so that

$$\operatorname{dist}_{L^2(K_j)}(v, V_j) \le c_{\mathrm{appr}} \frac{\operatorname{diam}(K_j)}{\sqrt[d]{k}} \|\nabla v\|_{L^2(K_j)} \qquad \text{for all } v \in Z_j \cap H^1(K_j).$$

Using the inequalities $\sqrt[d]{k} \ge \beta p$ and $\operatorname{diam}(K_j) = \operatorname{diam}(K) + 2r_j \le \operatorname{diam}(K) + 2\delta$ (see (5.46a)), we obtain

$$\operatorname{dist}_{L^2(K_j)}(v, V_j) \le c_{\mathrm{appr}} \frac{\operatorname{diam}(K) + 2\delta}{\beta p} \|\nabla v\|_{L^2(K_j)} \qquad \text{for all } v \in Z_j \cap H^1(K_j). \tag{5.47b}$$

Due to Remark 5.22, each $v \in Z_{j-1}$ restricted to K_j belongs to $Z_j \cap H^1(K_j)$. The estimates (5.47a,b) together with $\operatorname{diam}(K) \le \eta\delta$ shows

$$\operatorname{dist}_{L^2(K_j)}(v, V_j) \le (\eta + 2) \frac{2c_{\mathrm{appr}}\sqrt{\kappa_C}}{\beta} \|v\|_{L^2(K_{j-1})} \qquad \text{for all } v \in Z_{j-1}. \tag{5.47c}$$

In order to enforce $(\eta + 2) \frac{2c_{\mathrm{appr}}\sqrt{\kappa_C}}{\beta} = \varepsilon^{1/p}$, we choose

$$\beta := \beta_0 \varepsilon^{-1/p} \quad \text{with } \beta_0 := 2c_{\mathrm{appr}}\sqrt{\kappa_C}\,(\eta + 2)\,. \tag{5.47d}$$

e) (5.47c,d) shows that for each $u =: v_0 \in Z_0$ there is a $u_1 \in V_1 \subset Z_1$, so that $u|_{K_1} = v_0|_{K_1} = u_1 + v_1$ and

$$\|v_1\|_{L^2(K_1)} \le \varepsilon^{1/p} \|v_0\|_{L^2(K_0)}.$$

Analogously, for $v_1 \in Z_1$ there exists a $u_2 \in V_2 \subset Z_2$, so that $v_1|_{K_2} = u_2 + v_2$ and $\|v_2\|_{L^2(K_2)} \le \varepsilon^{1/p} \|v_1\|_{L^2(K_1)}$. By induction one constructs $u_j \in V_j$ $(1 \le j \le p)$, so that

$$u|_K = v_p + \sum_{j=1}^{p} u_j|_K \qquad \text{with } \|v_p\|_{L^2(K)} \le \varepsilon \|u\|_{L^2(K)}.$$

Since $u_j|_K \in V_j|_K$,

$$W := \operatorname{span}\{V_j|_K : j = 1, \dots, p\}$$

is the desired approximating subspace which ensures the estimate

$$\operatorname{dist}_{L^2(D_2)}(u, W) \le \varepsilon \|u\|_{L^2(K_0)} \underset{K_0 \subset D}{\le} \varepsilon \|u\|_{L^2(D)} \underset{u|_{D\setminus\Omega}=0}{=} \varepsilon \|u\|_{L^2(D\cap\Omega)}.$$

f) The dimension of W is bounded by $\sum_{j=1}^{p} \dim V_j = p\lceil (\beta p)^d \rceil \le p + \beta^d p^{d+1}$. The choice $p := \lceil \log \frac{1}{\varepsilon} \rceil$ yields $\varepsilon^{-1/p} = \mathrm{e}^{(\log \frac{1}{\varepsilon})/p} \le \mathrm{e}^1$ and the estimate

$$\dim W \le \lceil \log \tfrac{1}{\varepsilon} \rceil + \beta_0^d \mathrm{e}^d \lceil \log \tfrac{1}{\varepsilon} \rceil^{d+1}. \tag{5.47e}$$

This inequality together with $c_\eta := \beta_0 \mathrm{e}$ proves the assertion. ∎

Remark 5.25 *Lemma 5.24 describes the dimension $k := \dim W$ in dependence of the factor ε. The inverted relation shows the exponential decay*

$$\varepsilon = \varepsilon(k) \approx \exp\left(-c\, \sqrt[d+1]{k}\right) \qquad \textit{with } c \approx (c_\eta)^{-d/(d+1)}$$

(the equality $c = (c_\eta)^{-d/(d+1)}$ would hold, if on the right-hand side of (5.47e) the term $\lceil \log \frac{1}{\varepsilon} \rceil$ of lower order is missing, whereas $\varepsilon(k) \le \exp(-c\, \sqrt[d+1]{k})$ would hold, if $\lceil \log \frac{1}{\varepsilon} \rceil$ is replaced by $\log \frac{1}{\varepsilon}$).

For $x \in X \subset \Omega \subset \mathbb{R}^d$ Green's function $G(x, \cdot)$ is L-harmonic in $\Omega \setminus \overline{X}$, i.e., $G(x, \cdot) \in Z(\Omega \setminus \overline{X})$ and even $G(x, \cdot) \in Z(\mathbb{R}^d \setminus \overline{X})$ because of the continuation by zero.

Theorem 5.26 *Let $X \subset \Omega \subset \mathbb{R}^d$ and $K \subset \mathbb{R}^d$ be two domains with $K \cap \Omega \neq \emptyset$. Furthermore, let K be convex and satisfy*

$$\operatorname{diam}(K) \leq \eta \operatorname{dist}(X, K).$$

Then for any $\varepsilon \in (0,1)$ there exists a separable approximation

$$G_k(x,y) = \sum_{i=1}^{k} u_i(x) v_i(y) \qquad \text{with } k \leq k_\varepsilon = c_\eta^d \lceil \log \tfrac{1}{\varepsilon} \rceil^{d+1} + \lceil \log \tfrac{1}{\varepsilon} \rceil$$

(c_η defined in (5.45)) satisfying

$$\|G(x,\cdot) - G_k(x,\cdot)\|_{L^2(K)} \leq \varepsilon \|G(x,\cdot)\|_{L^2(D\cap\Omega)} \qquad \text{for all } x \in X, \tag{5.48}$$

where $D := \{y \in \mathbb{R}^d : \operatorname{dist}(y,K) < \operatorname{dist}(X,K)\}$.

Proof: Since $\operatorname{diam}(K) \leq \eta \operatorname{dist}(X,K) = \eta \operatorname{dist}(X, \partial D) \leq \eta \operatorname{dist}(X, \Gamma(D))$, Lemma 5.24 can be applied. Let $\{v_1, \ldots, v_k\}$ be a basis of the subspace $W \subset Z(K)$ from Lemma 5.24 with $k = \dim W \leq c_\eta^d \lceil \log \frac{1}{\varepsilon} \rceil^{d+1} + \lceil \log \frac{1}{\varepsilon} \rceil$. For all $x \in X$ the function $g_x := G(x,\cdot)$ belongs to $Z(D)$ because of $X \cap D = \emptyset$. Due to (5.44) we have $g_x = \hat{g}_x + r_x$ with $\hat{g}_x \in W$ and $\|r_x\|_{L^2(K)} \leq \varepsilon \|g_x\|_{L^2(D\cap\Omega)}$. The approximation $\hat{g}_x$ has a representation

$$\hat{g}_x = \sum_{i=1}^{k} u_i(x) v_i \tag{5.49}$$

with coefficients $u_i(x)$ depending on x. Since x varies in X, the coefficients u_i are functions defined on X. The function $G_k(x,y) := \sum_{i=1}^k u_i(x) v_i(y)$ satisfies the estimate (5.48). ∎

Remark 5.27 *Without loss of generality, $\{v_1, \ldots, v_k\}$ can be chosen as an* orthogonal basis *of W. Then the coefficients $u_i(x)$ in (5.49) are equal to the scalarproduct $(G(x,\cdot), v_i)_{L^2(K\cap\Omega)}$. This proves that the functions u_i satisfy the differential equation*

$$Lu_i = \begin{cases} v_i & \text{in } K \cap \Omega, \\ 0 & \text{otherwise} \end{cases}$$

with homogeneous Dirichlet boundary condition. In particular, the u_i are L-harmonic in $\Omega \backslash K$. Note that the u_i do not depend on the choice of the domain X.

Lemma 5.28 *Let X, K, D, and ε as in Theorem 5.26. Let $\mathcal{G}_{XK}$, $\mathcal{G}_{XD}$, and $\mathcal{G}_{k,XK}$ be the integral operators*

$$\begin{aligned} (\mathcal{G}_{XK} f)(x) &= \textstyle\int_Y G(x,y) f(y) \mathrm{d}y \qquad \text{for } x \in X, \\ (\mathcal{G}_{XD} f)(x) &= \textstyle\int_{D\cap\Omega} G(x,y) f(y) \mathrm{d}y \qquad \text{for } x \in X, \\ (\mathcal{G}_{k,XK})(x) &= \textstyle\int_Y G_k(x,y) f(y) \mathrm{d}y \qquad \text{for } x \in X, \end{aligned}$$

while $\mathcal{G}$ is the operator from (5.25). Then

$$\|\mathcal{G}_{XK} - \mathcal{G}_{k,XK}\|_{L^2(X) \leftarrow L^2(K\cap\Omega)} \leq \varepsilon \|\mathcal{G}_{XD}\|_{L^2(X) \leftarrow L^2(D\cap\Omega)} \leq \varepsilon \|\mathcal{G}\|_{L^2(\Omega) \leftarrow L^2(\Omega)}. \tag{5.50}$$

Proof: Let $\varphi \in L^2(X)$ be an arbitrary test function and $\Phi(y) := \int_X G(x,y)\varphi(x)\mathrm{d}x$ for $y \in D \cap \Omega$. Since $\Phi \in Z(D)$, we have again

$$\|\Phi - \Phi_k\|_{L^2(K\cap\Omega)} \leq \varepsilon \|\Phi\|_{L^2(D\cap\Omega)} \tag{5.51}$$

(proof as in Theorem 5.26 with the same W). As Φ_k is the projection of Φ to the subspace W, it follows that

$$\Phi_k(y) = \int_X G_k(x,y)\varphi(x)\mathrm{d}x = \sum_{i=1}^{k} \left(\int_X u_i(x)\varphi(x)\mathrm{d}x \right) v_i(y).$$

For all $\psi \in L^2(K \cap \Omega)$ we have

$$(\varphi, (\mathcal{G}_{XY} - \mathcal{G}_{k,XY})\,\psi)_{L^2(X)} = \int_{K\cap\Omega}\int_X (G(x,y) - G_k(x,y))\,\varphi(x)\psi(y)\mathrm{d}x\mathrm{d}y = (\Phi - \Phi_k, \psi)_{L^2(K\cap\Omega)}$$
$$\leq \|\Phi - \Phi_k\|_{L^2(K\cap\Omega)}\,\|\psi\|_{L^2(K\cap\Omega)} \underset{(5.51)}{\leq} \varepsilon\|\Phi\|_{L^2(D\cap\Omega)}\,\|\psi\|_{L^2(K\cap\Omega)}\,.$$

Φ can also be written as $\mathcal{G}^*_{XD}\varphi$ so that

$$\|\Phi\|_{L^2(D\cap\Omega)} \leq \|\mathcal{G}^*_{XD}\|_{L^2(D\cap\Omega)\leftarrow L^2(X)}\,\|\varphi\|_{L^2(X)} = \|\mathcal{G}_{XD}\|_{L^2(X)\leftarrow L^2(D\cap\Omega)}\,\|\varphi\|_{L^2(X)}$$

proves the first inequality in (5.50). The second inequality holds, since $\mathcal{G}_{XD}$ is a restriction of $\mathcal{G}$. ∎

We now show that the assumption in (5.31a-d) hold. The approximation (5.31a) corresponds to the representation in Theorem 5.26. The notation in (5.31a) indicates that the functions u_i, v_i depend on the dimension k.

If the minimum in (5.31b) is taken by $\operatorname{diam}(Y)$, we set $Y = K$ and the inequality in (5.31c) follows from (5.50). If, however, $\operatorname{diam}(X) \leq \eta \operatorname{dist}(X,Y)$ holds with a convex X, the same estimate can be proved by exploiting L-harmonicity with respect to the first argument: $G(\cdot, y) \in Z(X)$.

The sizes of the constants in (5.31d) result from Remark 5.25.

In the boundary element method (BEM) the fundamental solution S plays a central role. It is defined by the property

$$L_x S(x,y) = \delta(x-y) \qquad \text{for all } x, y \in \mathbb{R}^d.$$

We can apply Theorem 5.26 to S. The following corollary ensures that BEM matrices can be successfully represented in the hierarchical format.

Corollary 5.29 *Assume the existence of a fundamental solution S for the differential operator L. Let $X, Y \subset \mathbb{R}^d$ be two domains such that Y is convex and*

$$\operatorname{diam}(Y) \leq \eta \operatorname{dist}(X,Y).$$

Then for any $\varepsilon > 0$, there is a separable approximation

$$S_k(x,y) = \sum_{i=1}^{k} u_i(x)v_i(y) \quad \textit{with } k \leq k_\varepsilon = c_\eta^d \lceil \log \tfrac{1}{\varepsilon} \rceil^{d+1} + \lceil \log \tfrac{1}{\varepsilon} \rceil \qquad (c_\eta \textit{ as in (5.45)}),$$

so that

$$\|S(x,\cdot) - S_k(x,\cdot)\|_{L^2(Y)} \leq \varepsilon \|S(x,\cdot)\|_{L^2(D)} \qquad \textit{for all } x \in X$$

with $D := \{x \in \mathbb{R}^d : \operatorname{dist}(x,Y) < \operatorname{dist}(X,Y)\}$.

5.6 Implementation

The matrix G defined by (5.4) is sparse, so we should store it in a suitable format.

5.6.1 Sparse Matrices

One way of storing a sparse matrix efficiently is to use the *row-compressed representation*: For each row i, there are arrays containing the columns of non-zero entries and the corresponding coefficients.

Implementation 5.30 (`sparsematrix`) *The* `sparsematrix` *structure is defined as follows:*

```
typedef struct _sparsematrix sparsematrix;
typedef sparsematrix *psparsematrix;
struct _sparsematrix {
  int rows;
  int cols;
  int nz;
  int *row;
  int *col;
  double *coeff;
};
```

The fields `rows` *and* `cols` *contain the number of rows and the number of columns of the sparse matrix. The field* `nz` *contains the number of non-zero entries.*

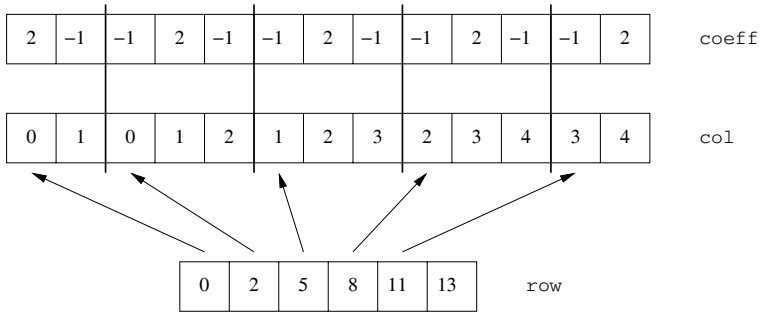
The columns indices for all non-zero entries are stored in the array `col`*, and the array* `coeff` *contains the corresponding coefficients. The array* `row` *contains the start and end indices of the parts of the* `col` *and* `coeff` *arrays corresponding to a certain row: For each row* `i`*, the entry* `row[i]` *is the first index in* `col` *and* `coeff` *corresponding to the* `i`*-th row of the matrix, and* `row[i+1]-1` *is the last index.*

Let us consider the matrix

$$G = \begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & 2 & -1 & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix}$$

as an example (in `C` notation, i.e., with indices $\{0, \ldots, 4\}$). The first row contains two non-zero entries, so we set `row[0]=0` and `row[1]=2`. The first non-zero entry is $G_{00} = 2$, therefore we set `col[0]=0` and `coeff[0]=2.0`. The second non-zero entry is $G_{01} = -1$, and we represent it by setting `col[1]=1` and `coeff[1]=-1.0`.

The second row contains three non-zero entries. We already have found `row[1]=2`, so we add three to get `row[2]=5`. The first non-zero entry $G_{10} = -1$ in this row is represented by `col[2]=0` and `coeff[2]=-1.0`. Repeating this construction for the remaining non-zero entries leads to the following representation:



The product of a vector v and a matrix described by a sparsematrix object can be computed by the following simple algorithm:

```
for(i=0; i<rows; i++) {
  sum = 0.0;
  for(j=row[i]; j<row[i+1]; j++)
    sum += coeff[j] * v[col[j]];
  w[i] = sum;
}
```

This and a number of other useful functions are already included in the library:

Implementation 5.31 (sparsematrix) *Once the structure of a sparse matrix (i.e., the arrays* row *and* col*) is fixed, we can use the functions*

```
void
clear_sparsematrix(psparsematrix sp);

void
addcoeff_sparsematrix(psparsematrix sp, int row, int col, double val);

void
setcoeff_sparsematrix(psparsematrix sp, int row, int col, double val);
```

to initialize the coefficients. clear_sparsematrix *sets all coefficients to zero,* addcoeff_sparsematrix *adds* val *to the coefficient in row* row *and column* col*, while* setcoeff_sparsematrix *sets the coefficient to* val *directly. The first two functions are useful in the context of finite-element discretisations, where the stiffness and mass matrices are usually assembled iteratively, while the third function was included for finite difference methods, where all matrix coefficients are known a priori.*

Of course, there are also the usual eval *methods*

```
void
eval_sparsematrix(pcsparsematrix sp, const double *v, double *w);

void
addeval_sparsematrix(pcsparsematrix sp, const double *v, double *w);

void
evaltrans_sparsematrix(pcsparsematrix sp, const double *v, double *w);

void
addevaltrans_sparsematrix(pcsparsematrix sp, const double *v, double *w);
```

that multiply a sparse matrix or its transposed by a vector and add the result to another vector or overwrite it.

A sparsematrix *is not a* supermatrix*. Therefore we have to convert it before we can apply $\mathcal{H}$-matrix arithmetics. Due to Lemma 5.1, this is a simple task that is accomplished by the function*

```
void
convertsparse2_supermatrix(pcsparsematrix sp, psupermatrix s);
```

Defining the proper structure for a sparse matrix can be complicated, since it involves counting the number of non-zero entries and enumerating them correctly. Therefore we introduce the auxiliary structure sparsefactory that handles all the bookkeeping:

Implementation 5.32 (`sparsefactory`) *A* `sparsefactory` *object is used to describe only the structure of a sparse matrix, i.e., the distribution of non-zero entries. It is created by a call to*

```
void
new_sparsefactory(int rows, int cols);
```

giving the number `rows` *of rows and* `cols` *of columns of the sparse matrix. A new* `sparsefactory` *contains no non-zero entries, i.e., it describes the zero matrix. We can add non-zero entries using the function*

```
void
addnz_sparsefactory(psparsefactory fac, int row, int col);
```

where `row` *and* `col` *are the row and column of a new non-zero entry. This function call will only change the* `sparsefactory` *if the entry is currently not in the list of non-zeros, i.e., it is safe to call it more than once for the same entry.*

After the entire structure of the matrix has been described by calls to `addnz_sparsefactory`*, we can use the* `sparsefactory` *to create a matching* `sparsematrix`*:*

```
psparsematrix
new_sparsematrix(psparsefactory fac);
```

Let us now apply these techniques to Example 2.12: we use piecewise linear basis functions, i.e., an entry G_{ij} of the stiffness matrix (5.4) can be non-zero only if there is at least one triangle in the grid that lies in the support of both φ_i and φ_j. This information is sufficient to create the `sparsematrix` structure:

```
idx2dof = ct->idx2dof;

fac = new_sparsefactory(n, n);
for(k=0; k<triangles; k++)
  for(i=0; i<3; i++) {
    ii = idx2dof[t[k][i]];
    if(ii >= 0)
      for(j=0; j<3; j++) {
        jj = idx2dof[t[k][j]];
        if(jj >= 0)
          addnz_sparsefactory(fac, ii, jj);
      }
  }

sp = new_sparsematrix(fac);
del_sparsefactory(fac);
```

Since the `sparsematrix` structure should be compatible with a `supermatrix` structure, we have to take the permutation of the index set into account. The code in Example 2.12 stores the mapping from grid indices to degrees of freedom in the array `ct->idx2dof`, and our code fragment uses this array to convert the indices of the triangles vertices `t[k][i]` into degrees of freedom. If `t[k][i]` does not correspond to a degree of freedom, i.e., if it is a boundary node, `idx2dof[t[k][i]]` has the value `-1` and we drop this entry from the sparse matrix.

5.6.2 Assembly of Stiffness and Mass Matrices

We have seen how a `sparsematrix` object matching the structure of a grid and a set of basis functions can be constructed. Now, we will fill the sparse matrix with coefficients corresponding to the partial differential equation.

The matrix coefficients are defined by (5.4). As in Example 2.12, we consider only domains that correspond to compatible triangulations, i.e., there is a family $(\tau_\iota)_{\iota=0}^{T-1}$ of triangles such that

$$\overline{\Omega} = \bigcup_{\iota=0}^{T-1} \overline{\tau}_\iota$$

holds and that two different triangles τ_ι, τ_κ are either disjoint, share a common vertex or a common edge.

The discrete space V_n of piecewise linear functions on the triangulation $(\tau_\iota)_{\iota=0}^{T-1}$ is defined by

$$V_n := \{u \in C(\Omega) : u|_{\tau_\iota} \text{ is affine for all } \iota \in \{0, \ldots, T-1\} \text{ and } u|_\Gamma = 0\} \subseteq H_0^1(\Omega)$$

Let $u \in V_n$. Since u is affine on each τ_ι, its restriction $u|_{\tau_\iota}$ it is uniquely defined by the values in the vertices of τ_ι. Since u is continuous, it is uniquely defined by its values of u in all vertices of the triangulation, which implies that the dimension n of V_n has to be the number of interior vertices in the triangulation. We denote the interior vertices by $(v_i)_{i\in\mathcal{I}}$ and define the basis functions $(\varphi_i)_{i\in\mathcal{I}}$ by

$$\varphi_i(v_j) = \delta_{ij}$$

for all $j \in \mathcal{I}$. Due to (5.4), the entry G_{ij} is given by

$$G_{ij} = \int_\Omega \langle \operatorname{grad} \varphi_i(x), \operatorname{grad} \varphi_j(x) \rangle \, \mathrm{d}x = \sum_{\iota=0}^{T-1} \int_{\tau_\iota} \langle \operatorname{grad} \varphi_i(x), \operatorname{grad} \varphi_j(x) \rangle \, \mathrm{d}x,$$

so we could build the matrix using the code fragment

```
for(ii=0; ii<n; ii++)
  for(jj=0; jj<n; jj++) {
    sum = 0.0;
    for(k=0; k<triangles; k++)
      if(in_support(k, ii) && in_support(k, jj))
        addcoeff_sparsematrix(G, ii, jj, integrate_G(k, ii, jj));
  }
```

This is a very inefficient approach, since all entries of the stiffness matrix G are considered at least once, i.e., we will have a quadratic complexity in the number n of degrees of freedom, which is clearly inacceptable for interesting problem sizes.

A far more efficient approach switches the ordering of the loops: the outer loop passes through all triangles, the inner loops consider *all* contributions a given triangle τ_ι makes to the matrix G. According to our definition of the basis functions, only the three basis functions corresponding to the vertices of the triangle can differ from zero in τ_ι. Therefore we only have to take pairs of these three basis functions into account, i.e., we require two nested loops passing each through the vertices of the current triangle. This leads to the following more efficient algorithm:

```
for(k=0; k<triangles; k++)
  for(i=0; i<3; i++) {
    ii = idx2dof[t[k][i]];
    if(ii >= 0)
      for(j=0; j<3; j++) {
        jj = idx2dof[t[k][j]];
        if(jj >= 0)
          addcoeff_sparsematrix(G, ii, jj, integrate_G(k, ii, jj));
      }
  }
```

In order to evaluate the individual integrals (5.4), we need the gradients of the basis functions. For piecewise linear basis functions, their computation is very simple: let $i, j, k \in \{0, \ldots, n-1\}$ be such that v_i, v_j, v_k are the vertices of a triangle τ_ι in the triangulation. The function

$$\varphi_{i,\tau_\iota}(x) := \frac{\det(x - v_j, v_k - v_j)}{\det(v_i - v_j, v_k - v_j)}$$

if affine and obviously satisfies $\varphi_{i,\tau_\iota}(v_i) = 1$, $\varphi_{i,\tau_\iota}(v_j) = \varphi_{i,\tau_\iota}(v_k) = 0$. This implies $\varphi_i|_{\tau_\iota} = \varphi_{i,\tau_\iota}$, and we can use this representation to compute the gradients of basis functions:

```
det = ((p[t[k][0]][0] - p[t[k][2]][0]) *
       (p[t[k][1]][1] - p[t[k][2]][1]) -
       (p[t[k][1]][0] - p[t[k][2]][0]) *
       (p[t[k][0]][1] - p[t[k][2]][1]));

for(i=0; i<3; i++) {
  g[i][0] = (p[t[k][(i+1)%3]][1] - p[t[k][(i+2)%3]][1]) / det;
  g[i][1] = (p[t[k][(i+2)%3]][0] - p[t[k][(i+1)%3]][0]) / det;
}
```

We can combine this fragment with the loop over all triangles in order to complete the algorithm for assembling the stiffness matrix for the Laplace operator:

```
for(k=0; k<triangles; k++) {
  det = ((p[t[k][0]][0] - p[t[k][2]][0]) *
         (p[t[k][1]][1] - p[t[k][2]][1]) -
         (p[t[k][1]][0] - p[t[k][2]][0]) *
         (p[t[k][0]][1] - p[t[k][2]][1]));
  for(i=0; i<3; i++) {
    g[i][0] = (p[t[k][(i+1)%3]][1] - p[t[k][(i+2)%3]][1]) / det;
    g[i][1] = (p[t[k][(i+2)%3]][0] - p[t[k][(i+1)%3]][0]) / det;
  }
  area = 0.5 * fabs(det);

  for(i=0; i<3; i++) {
    ii = idx2dof[t[k][i]];
    if(ii >= 0)
      for(j=0; j<3; j++) {
        jj = idx2dof[t[k][j]];
        if(jj >= 0) {
          val = area * (g[i][0] * g[j][0] + g[i][1] * g[j][1]);
          addcoeff_sparsematrix(G, ii, jj, val);
        }
      }
  }
}
```

Chapter 6

Arithmetics of Hierarchical Matrices

In this chapter we will explain the algorithms that perform the addition and multiplication in the hierarchical matrix format efficiently. Based upon these basic linear algebra subroutines, we can define algorithms that compute an approximate inverse, LU-decompostion or Cholesky decomposition. The actual proof for the efficiency, namely the complexity estimates, are postponed to the next Chapter 7. The basic idea for the $\mathcal{H}$-matrix arithmetics is formulated in [40] and a general approach is contained in [27] (german) and [29] (english).

6.1 Arithmetics in the `rkmatrix` Representation

Since the basic building blocks of $\mathcal{H}$-matrices are matrices in `fullmatrix` and `rkmatrix` representation we will first explain how the arithmetic operations $+, \cdot$ can be performed efficiently for matrices in `rkmatrix` format - for the `fullmatrix` format this is obvious (and already implemented in BLAS or LAPACK).

First, we have to introduce the set of matrices of rank at most k. These are the matrices that can be represented in the `rkmatrix` format. Afterwards, we will modify the `rkmatrix` implementation.

Implementation 6.1 (`rkmatrix`) *The* `rkmatrix` *representation is implemented in the* C *programming language as follows:*

```
typedef struct _rkmatrix rkmatrix;
typedef rkmatrix *prkmatrix;

struct _rkmatrix {
  int k;
  int kt;
  int rows;
  int cols;
  double* a;
  double* b;
};
```

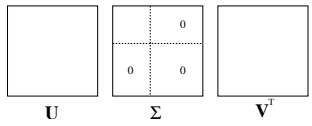
The description is the same as in the previous Implementation 1.13.

The current rank `kt` resembles the fact that a matrix in $\mathcal{R}(k,n,m)$ can have a rank kt smaller than k. From Lemma 5.1 we know that all `rkmatrix` blocks in the $\mathcal{H}$-matrix representation of the stiffness matrix are of rank 0 while the maximal allowed rank for the formatted arithmetics is $k > 0$. In the algorithms we want to exploit $kt < k$ whenever possible.

6.1.1 Reduced Singular Value Decomposition (rSVD)

Definition 6.2 (SVD and rSVD) *Let $M \in \mathcal{R}(k, n, m)$. A singular value decomposition (SVD) of M is a factorisation of the form*

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



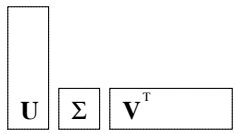
with unitary matrices $U \in \mathbb{R}^{n\times n}$, $V \in \mathbb{R}^{m\times m}$ and a diagonal matrix $\Sigma \in \mathbb{R}^{n\times m}$, where the diagonal entries are

$$\Sigma_{11} \geq \Sigma_{22} \geq \ldots \geq \Sigma_{kk} \quad \geq \quad \Sigma_{k+1,k+1} = \ldots = \Sigma_{\min\{n,m\},\min\{n,m\}} = 0.$$

The diagonal entries of Σ are called the singular values of M.

A reduced singular value decomposition (rSVD) of M is a factorisation of the form

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



with matrices $U \in \mathbb{R}^{n\times k}$, $V \in \mathbb{R}^{m\times k}$ that have orthonormal columns and a diagonal matrix $\Sigma \in \mathbb{R}^{k\times k}$, where the diagonal entries are

$$\Sigma_{11} \geq \Sigma_{22} \geq \ldots \geq \Sigma_{kk} \quad > 0.$$

Remark 6.3 *1. A (reduced) SVD is not unique.*

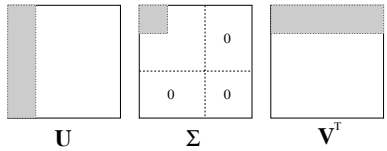
2. *If the singular values of M are all different then the reduced singular value decomposition is unique up to scaling of the columns of U, V by -1.*
3. *A SVD of M yields a rSVD by discarding the columns $> k$ of U, V and the columns and rows $> k$ of Σ.*

A SVD of a general matrix (`fullmatrix`) can be computed by the standard LAPACK subroutine `dgesvd` within complexity $\mathcal{O}\left(\min(n,m)\max(n,m)^2\right)$ (see [21]). After the next Lemma we will explain how to compute a rSVD of an `rkmatrix` in $\mathcal{O}\left(k^2\max(n,m)\right)$ complexity.

The singular value decomposition is a representation of a matrix (in factorised form) and the reduced singular value decomposition is similar to the `rkmatrix` format. The reason why the SVD is of interest is given in the next Lemma.

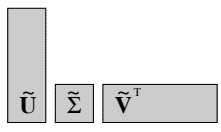
Lemma 6.4 (Best Approximation with Fixed Rank) *Let*

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



be a SVD of $M \in \mathbb{R}^{n\times m}$. Let

$$\tilde{M} := \tilde{U}\tilde{\Sigma}\tilde{V}^T$$



with matrices

$$\tilde{U} := U|_{n\times k}, \quad \tilde{\Sigma} := \operatorname{diag}(\Sigma_{1,1}, \ldots, \Sigma_{k,k}), \quad \tilde{V} := V|_{m\times k}.$$

Then $\tilde{M}$ is a best approximation to M in the sense that

$$\|M - \tilde{M}\| = \min_{R\in\mathcal{R}(k,n,m)} \|M - R\|, \|M - \tilde{M}\|_2 = \Sigma_{k+1,k+1}, \|M - \tilde{M}\|_F = \sqrt{\sum_{i=k+1}^{\min(n,m)} \Sigma_{i,i}^2}.$$

holds in the Frobenius and spectral norm.

Proof: For the spectral norm the proof is contained in [21]. The extension to the Frobenius norm can be achieved by induction as an exercise. ■

Let $M = AB^T \in \mathbb{R}^{n\times m}$ be a matrix in `rkmatrix` representation. A rSVD $M = U\Sigma V^T$ can be computed efficiently in three steps:

1. Compute (reduced) QR-factorisations of A, B: $A = Q_A R_A$, $B = Q_B R_B$ with matrices $Q_A \in \mathbb{R}^{n\times k}$, $Q_B \in \mathbb{R}^{m\times k}$, $R_A, R_B \in \mathbb{R}^{k\times k}$. `dgeqrf`→ complexity $\mathcal{O}((n+m)k^2)$
2. Compute a rSVD of $R_A R_B^T = U'\Sigma(V')^T$. `dgesvd`→ complexity $\mathcal{O}(k^3)$
3. Compute $U := Q_A U'$, $V := Q_B V'$. `dgemm`→ complexity $\mathcal{O}((n+m)k^2)$

Implementation 6.5 (rSVD) *The implemementation of the rSVD in the* C *programming language for the* `rkmatrix` *format might look as follows:*

```
void
rsvd_rkmatrix(prkmatrix r, double *u, double* s, double *v){
  double *u_a, *v_a, *u_b, *v_b, *usv;
  int kt = r->kt, rows = r->rows, cols = r->cols;

  ... allocate u_a,v_a,u_b,v_b,usv ...

  qr_factorisation(r->a,rows,kt,u_a,v_a);      /* r->a =: u_a*v_a   */
  qr_factorisation(r->b,cols,kt,u_b,v_b);      /* r->b =: u_b*v_b   */

  multrans2_lapack(kt, kt, kt, v_a, v_b, usv); /* usv := v_a*v_b'   */
  svd_factorisation(usv, u_s, s, v_s);         /* usv =: u_s*s*v_s' */

  mul_lapack(rows, kt, kt, u_a, u_s, u);       /* u := u_a*u_s      */
  mul_lapack(cols, kt, kt, v_b, v_s, v);       /* v := v_b*v_s      */

  ... deallocate u_a,v_a,u_b,v_b,usv ...
}
```

The procedure `rsvd_rkmatrix` enables us to compute a rSVD of an `rkmatrix` in $\mathcal{O}\left(k^2(n+m)\right)$ complexity. According to Lemma 6.4 the rSVD representation can be used to derive a best approximation with fixed rank.

Definition 6.6 (Truncation $\mathcal{T}_k$) *Let $M \in \mathbb{R}^{n\times m}$ and $k \in \mathbb{N}$. We define the truncation operator $\mathcal{T}_k : \mathbb{R}^{n\times m} \to \mathcal{R}(k,n,m)$, $M \mapsto \tilde{M}$, where $\tilde{M}$ is a best approximation of M in the set $\mathcal{R}(k,n,m)$ (not necessarily unique). The matrix $\tilde{M}$ is called a "truncation of M to rank k".*

The truncation operator $\mathcal{T}_k$ produces an approximation to a given matrix in the set $\mathcal{R}(k,n,m)$. The approximation quality might be arbitrarily bad, although it is a best approximation within the set. An alternative truncation operator is defined by a fixed accuracy that has to be achieved. The rank necessary to reach the accuracy is chosen automatically.

NumCSE exercise: implementation of \epsilon truncation

Definition 6.7 (Truncation $\mathcal{T}_\varepsilon$) *Let $M \in \mathbb{R}^{n\times m}$ and $\varepsilon > 0$. We define the truncation operator $\mathcal{T}_\varepsilon : \mathbb{R}^{n\times m} \to \mathcal{R}(k,n,m)$, $M \mapsto \tilde{M}$, where $\tilde{M}$ is a best approximation of M in the set $\mathcal{R}(k,n,m)$ and*

$$k := \min\{\tilde{k} \in \mathbb{N}_0 \mid \exists \tilde{M} \in \mathcal{R}(\tilde{k},n,m) : \|M - \tilde{M}\| \le \varepsilon\|M\|\}.$$

The matrix $\tilde{M}$ is called an "adaptive truncation of M with accuracy ε". $\mathcal{T}_\varepsilon^{\text{abs}}$ is the respective truncation operator with absolute truncation error ε.

From Lemma 6.4 we know how to determine the necessary rank for an adaptive truncation: compute a rSVD and discard singular values (starting from the smallest) as long as the relative or absolute prescribed accuracy is met.

6.1.2 Formatted `rkmatrix` Arithmetics

The set $\mathcal{R}(k,n,m)$ is *not* a linear subspace of the $n \times m$ matrices, because it is not closed with respect to the addition. Instead, the set has properties of an ideal.

Lemma 6.8 (Multiplication) *Let $R \in \mathcal{R}(k,n,m)$, $N \in \mathbb{R}^{n'\times n}$ and $M \in \mathbb{R}^{m\times m'}$. Then*

$$NR \in \mathcal{R}(k,n',m), \qquad RM \in \mathcal{R}(k,n,m').$$

Proof: If $R = AB^T$ then $NR = (NA)B^T$ and $RM = A(M^TB)^T$. ■

Lemma 6.9 (Addition) *Let $R_1, R_2 \in \mathcal{R}(k,n,m)$. Then $R_1 + R_2 \in \mathcal{R}(2k,n,m)$.*

Proof: If $R_1 = AB^T$ and $R_2 = CD^T$ then $R_1 + R_2 = AB^T + CD^T = \underbrace{\begin{bmatrix} A & C \end{bmatrix}}_{n\times 2k}\underbrace{\begin{bmatrix} B & D \end{bmatrix}^T}_{2k\times m}$. ■

The addition of two matrices in `rkmatrix` format of rank k_1 and k_2 yields a matrix in `rkmatrix` format of rank $k_1 + k_2$ without performing arithmetic operations (see the previous proof). The *formatted* sum is then defined as a best approximation of rank at most k, where k is either fixed (fixed rank addition) or chosen automatically so that a given approximation quality is achieved (adaptive addition).

Definition 6.10 (Formatted `rkmatrix` Addition) *The* formatted *addition (fixed rank k or adaptive with accuracy ε) is defined as*

$$A \oplus_k B := \mathcal{T}_k(A+B), \quad A \oplus_\varepsilon B := \mathcal{T}_\varepsilon(A+B), \qquad A, B \in \mathbb{R}^{n\times m}.$$

In the following implementation of the formatted `rkmatrix` addition we have combined the fixed rank and adaptive truncation.

Implementation 6.11 (Formatted `rkmatrix` Addition) *The structure* `truncation_control` *contains the information about the sufficient relative truncation error* `rel_eps`*, the sufficient absolute truncation error* `abs_eps` *and a flag* `adaptive` *that tells us wether we use the fixed rank truncation (*`adaptive`$= 0$*) or the adaptive arithmetic (*`adaptive`$= 1$*). This information is stored in the struct* `truncation_control`*:*

```
typedef struct _truncation_control truncation_control;
typedef truncation_control *ptruncation_control;

struct _truncation_control {
  double rel_eps;
  double abs_eps;
  int adaptive;
};
```

Each `rkmatrix r` *stores a pointer* `r->tc` *to a struct* `truncation_control` *which is by default* `NULL`*, i.e., we use by default the fixed rank arithmetic. If* `r->tc!=0` *and* `r->tc->adaptive`$= 1$ *then the rank* k *for the representation of the target matrix is determined by*

$$k := \min\{\tilde{k} \in \mathbb{N}_0 \mid \exists \tilde{M} \in \mathcal{R}(\tilde{k}, n, m) : \|M - \tilde{M}\| \leq \texttt{rel\_eps}\|M\| \text{ or } \|M - \tilde{M}\| \leq \texttt{abs\_eps}\}.$$

The structure of the target matrix has to be extended to allow the necessary rank k *(which is a priori not known).*

The implementation of the formatted addition in the C *programming language for the* `rkmatrix` *format is done as follows:*

```
void
add_rkmatrix(prkmatrix c, prkmatrix a, prkmatrix b){
  prkmatrix a_plus_b;
  double *u, *s, *v;
  int i, j, n = c->rows, m = c->cols, kt = a->kt + b->kt;

  a_plus_b = new_rkmatrix(kt,n,m);
  u = (double*) malloc(kt*n*sizeof(double));
  v = (double*) malloc(kt*m*sizeof(double));
  s = (double*) malloc(kt*sizeof(double));

  for(i=0; i<(a->kt)*n; i++) a_plus_b->a[i] = a->a[i];
  for(i=0; i<(a->kt)*m; i++) a_plus_b->b[i] = a->b[i];
  for(i=0; i<(b->kt)*n; i++) a_plus_b->a[i+(a->kt)*n] = b->a[i];
  for(i=0; i<(b->kt)*m; i++) a_plus_b->b[i+(a->kt)*m] = b->b[i];

  rsvd_rkmatrix(a_plus_b, u, s, v);

  if(c->tc && c->tc->adaptive){
     for(i=0; i<kt && s[i]>c->tc->rel_eps*s[0] && s[i]>c->tc->abs_eps; i++)
     if(i>c->k) reallocstructure_rkmatrix(c,i);
  }else{
     for(i=0; i<kt && i<c->k; i++);
  }
  c->kt = i;
  for(i=0; i<c->kt*n; i++) c->a[i] = u[i];
  for(i=0; i<m; i++)
     for(j=0; j<c->kt; j++)
        c->b[i+j*m] = v[i+j*m] * s[j];

  free(s); free(u); free(v);
  del_rkmatrix(a_plus_b);
}
```

Later, we will always use the procedure `addparts2_rkmatrix(r,nr,rk_no,rk_mo,rk_r)` that computes $\mathcal{T}_k(R)$ or $\mathcal{T}_\varepsilon(R)$ for the matrix R (cf. Figure 6.1) defined by

$$\begin{aligned} R &:= \texttt{r} + \sum_{l=0}^{\texttt{nr}} R_l, \\ (R_l)_{i,j} &:= \begin{cases} \texttt{rk\_r}_{i_l,j_l} & 0 \leq i_l < \texttt{rk\_r[l]->rows},\ 0 \leq j_l < \texttt{rk\_r[l]->cols}, \\ 0 & \text{otherwise.} \end{cases} \\ & \quad i_l := i - \texttt{rk\_no[l]}, \quad j_l := j - \texttt{rk\_mo[l]}. \end{aligned}$$

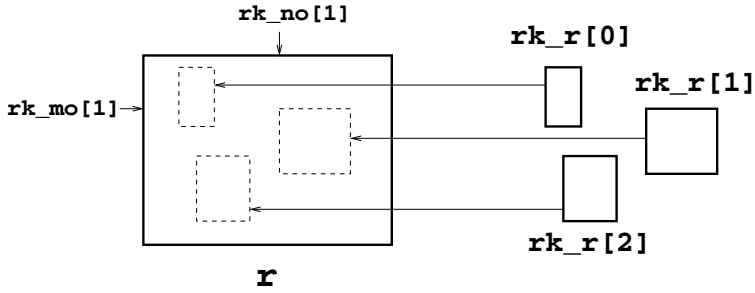


Figure 6.1: The matrix R is the sum of `r` and the by zero extended matrices `rk_r[1]`.

The standard addition $\mathcal{T}_k(A+B)$ is given by `nr` $:= 2$, `r` $:= 0$, `rk_r[0]` $:= A$, `rk_r[1]` $:= B$ and offsets `rk_no[1]` $=$ `rk_mo[1]` $= 0$. The implementation of `addparts2_rkmatrix(r,nr,rk_no,rk_mo,rk_r)` is similar to Implementation 6.11.

6.2 Arithmetics in the $\mathcal{H}$-Matrix Format

6.2.1 Addition and Multiplication

In Definitions 6.6 and 6.7, we have defined the truncation operator $\mathcal{T}_k$ to the set $\mathcal{R}(k,n,m)$. The extension to $\mathcal{H}$-matrices is given below.

Definition 6.12 (Truncation $\mathcal{T}_k$) *Let $T_{\mathcal{I}\times\mathcal{I}}$ be a block cluster tree. We define the truncation operator*

$$\mathcal{T}_k : \mathbb{R}^{n\times m} \to \mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, k), \quad M \mapsto \tilde{M}$$

blockwise for all leaves $t\times s \in T_{\mathcal{I}\times\mathcal{I}}$ by

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

Lemma 6.13 *The operator $\mathcal{T}_k$ maps a matrix $M \in \mathbb{R}^{\mathcal{I}\times\mathcal{I}}$ to a best approximation $\tilde{M} \in \mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, k)$ with respect to the Frobenius norm:*

$$\|M - \tilde{M}\|_F = \min_{M'\in\mathcal{H}(T_{\mathcal{I}\times\mathcal{I}},k)} \|M - M'\|_F$$

Proof: Exercise ■

The truncation operator can be used to define the formatted matrix operations as follows, cf. Algorithm 4.

Definition 6.14 (Formatted Addition) *Let $T_{\mathcal{I}\times\mathcal{I}}$ denote a block cluster tree and $k \in \mathbb{N}$. We define the formatted addition of $\mathcal{H}$-matrices $A, B \in \mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, k)$ by*

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

If the rank k under consideration is not evident then we write $\oplus_k$ instead of $\oplus$.

Definition 6.15 (Formatted Multiplication) *Let $T_{\mathcal{I}\times\mathcal{I}}$ be a block cluster tree and let $k \in \mathbb{N}$. We define the formatted multiplication of $\mathcal{H}$-matrices $A, B \in \mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, k)$ by*

$$A \odot B := \mathcal{T}_k(A\cdot B).$$

Algorithm 4

procedure AddH-Matrix(H-Matrices A,B,C);
for $t \times s \in \mathcal{L}(T_{\mathcal{I}\times\mathcal{I}})$ **do**
`add_rkmatrix(`$C|_{\hat{t}\times\hat{s}}$`,` $A|_{\hat{t}\times\hat{s}}$`,` $B|_{\hat{t}\times\hat{s}}$`);`
end for

Formally it is easy to write "$A \cdot B$", but in practice the structure of the matrix product $A \cdot B$ can become rather complicated. This is different from the addition where the structure of the sum $A + B$ retains the structure of A and B. To illustrate the complications, we take a look at two typical examples. After some necessary Definitions and Lemmata we finally explain how to compute the $\mathcal{H}$-matrix product efficiently.

Example 6.16 (Multiple Blocks per Row) *We consider $m \times m$ block matrices in $\mathbb{R}^{n\times n}$:*

$$A = \begin{bmatrix} A_{11} & \cdots & A_{1m} \\ \vdots & \ddots & \vdots \\ A_{m1} & \cdots & A_{mm} \end{bmatrix}, \qquad B = \begin{bmatrix} B_{11} & \cdots & B_{1m} \\ \vdots & \ddots & \vdots \\ B_{m1} & \cdots & B_{mm} \end{bmatrix}.$$

In the sum $A + B$ only one addition per block occurs:

$$(A + B)_{ij} = A_{ij} + B_{ij}$$

In the product $A \cdot B$ multiple terms appear:

$$(A \cdot B)_{ij} = \sum_{l=1}^{m} A_{il} \cdot B_{lj}$$

The truncation $\mathcal{T}_k$ of the sum over m addends is much more expensive than m times the truncation of two addends. However, the latter will not necessarily yield a best approximation.

Definition 6.17 (Fast Truncation $\mathcal{T}'_k$) *Let $T_{\mathcal{I}\times\mathcal{I}}$ be a block cluster tree and $q \in \mathbb{N}_{>1}$. We define the fast truncation operator*

$$\mathcal{T}'_k : \mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, qk) \to \mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, k), \quad M \mapsto \tilde{M}$$

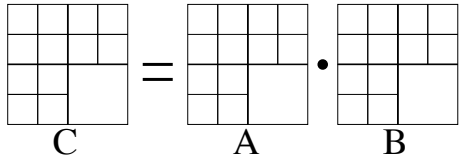
by $q-1$ times calling the truncation for a matrix with blockwise rank $2k$: let $M = \sum_{i=1}^{q} M_i$ be a decomposition of M into q matrices $M_i \in \mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, k)$. Then we define

$$\begin{aligned} \tilde{M}_1 &:= M_1, \\ \tilde{M}_i &:= \mathcal{T}_k(M_i + \tilde{M}_{i-1}), \quad i = 2, \ldots, q, \\ \tilde{M} &:= \tilde{M}_q. \end{aligned}$$

Example 6.18 (Different Matrix Formats) *We consider 2×2 block matrices in $\mathbb{R}^{n\times n}$,*

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \qquad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}, \qquad C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix},$$

and assume that all submatrices A_{ij}, B_{ij}, C_{ij} are again 2×2 block matrices consisting of matrices in `supermatrix` *format except the lower right blocks A_{22}, B_{22}, C_{22}, which belong to $\mathcal{R}(k, n_2, m_2)$. The product $A \cdot B$ is to be truncated to the format of C:*



In the lower right block of C we have to perform the truncation

$$C_{22} = \mathcal{T}_k\left(A_{21}B_{12} + A_{22}B_{22}\right)$$

to the `rkmatrix` *format. The product $A_{21}B_{12}$ is not contained in $\mathcal{R}(k, n_2, m_2)$ which poses a problem: we have to truncate a hierarchical matrix (*`supermatrix`*) to the* `rkmatrix` *format. To do this efficiently, we will not compute a best approximation but something close to it.*

Definition 6.19 (Levels of the Tree) *Let T be a tree. We define the levels $\ell \in \mathbb{N}_0$ of T by*

$$T^{(\ell)} := \{t \in V(T) \mid \operatorname{level}(t) = \ell\}.$$

On each level $\ell \in \mathbb{N}_0$ we define

$$\mathcal{L}(T, \ell) := T^{(\ell)} \cap \mathcal{L}(T).$$

Lemma 6.20 *For any cluster tree $T_\mathcal{I}$ and $\ell \in \mathbb{N}_0$ there holds*

$$\mathcal{I} = \dot{\bigcup}_{t \in L_\ell(T_\mathcal{I})} \hat{t}, \qquad L_\ell(T_\mathcal{I}) := T_\mathcal{I}^{(\ell)} \cup \mathcal{L}(T_\mathcal{I}, \ell - 1) \cup \ldots \cup \mathcal{L}(T_\mathcal{I}, 0).$$

Proof: We start by proving $\mathcal{I} = \bigcup_{t \in L_\ell} \hat{t}$. Let $i \in \mathcal{I}$. According to Lemma 2.7, there is a leaf cluster $t \in \mathcal{L}(T_\mathcal{I})$ with $i \in \hat{t}$. If $\operatorname{level}(t) < \ell$, we have $t \in L_\ell$. If $\operatorname{level}(t) \geq \ell$, let $m := \operatorname{level}(t)$, and let $t_0, \ldots, t_m$ be the sequence of ancestors of t. Due to $\ell \leq m$, $t^* := t_\ell$ is well-defined, satisfies $\operatorname{level}(t^*) = \ell$, i.e., $t^* \in T_\mathcal{I}^{(\ell)}$, and due to $t \in \operatorname{sons}^*(t^*)$, we have $i \in \hat{t} \subseteq \hat{t^*}$, so we have found $t^* \in L_\ell$ with $i \in \hat{t^*}$.

Now we prove that the elements of L_ℓ correspond to disjoint index sets. Let $t, s \in L_\ell$ with $\hat{t} \cap \hat{s} \neq \emptyset$. Without loss of generality, we assume $\operatorname{level}(t) \leq \operatorname{level}(s)$, and Lemma 2.7 yields $s \in \operatorname{sons}^*(t)$. If t is a leaf, this implies $s = t$. If t is not a leaf, the definition of L_ℓ implies $\operatorname{level}(t) = \ell$. Due to $s \in L_\ell$, we have $\operatorname{level}(s) \leq \ell$, and $\ell = \operatorname{level}(t) \leq \operatorname{level}(s) \leq \ell$ yields $\operatorname{level}(s) = \ell = \operatorname{level}(t)$. Using the first statement of Lemma 2.7, we can conclude $s = t$. ∎

Definition 6.21 (Hierarchical Approximation) *Let $T_{\mathcal{I}\times\mathcal{I}}$ be a block cluster tree and $p := \operatorname{depth}(T_{\mathcal{I}\times\mathcal{I}})$. For each $M \in \mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, k)$ we define the hierarchical approximation $M_\mathcal{H}$ of M in $p+1$ steps (see Figure 6.2) using the notation $L_\ell(T_{\mathcal{I}\times\mathcal{I}})$ from Lemma 6.20 by*

$$\begin{aligned}
M_p|_{\hat{t}\times\hat{s}} &:= \begin{cases} \mathcal{T}_k(M|_{\hat{t}\times\hat{s}}) & t \times s \in \mathcal{L}(T_{\mathcal{I}\times\mathcal{I}}, p), \\ M|_{\hat{t}\times\hat{s}} & t \times s \in L_p(T_{\mathcal{I}\times\mathcal{I}}) \setminus \mathcal{L}(T_{\mathcal{I}\times\mathcal{I}}, p) \end{cases}, \\
M_{\ell-1}|_{\hat{t}\times\hat{s}} &:= \begin{cases} \mathcal{T}_k(M_\ell|_{\hat{t}\times\hat{s}}) & t \times s \in \mathcal{L}(T_{\mathcal{I}\times\mathcal{I}}, \ell - 1), \\ M_\ell|_{\hat{t}\times\hat{s}} & t \times s \in L_{\ell-1}(T_{\mathcal{I}\times\mathcal{I}}) \setminus \mathcal{L}(T_{\mathcal{I}\times\mathcal{I}}, \ell - 1) \end{cases} \qquad \ell = p, \ldots, 1, \\
M_\mathcal{H} &:= M_0.
\end{aligned}$$

Lemma 6.22 (Hierarchical Approximation Error) *Let $T_{\mathcal{I}\times\mathcal{I}}$ be a block cluster tree, $p := \operatorname{depth}(T_{\mathcal{I}\times\mathcal{I}})$, $M \in \mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, k)$ and let $M_\mathcal{H}$ denote the hierarchical approximation of M. Then*

$$\|M - M_\mathcal{H}\|_F \leq (2^{p+1} + 1)\|M - \mathcal{T}_k(M)\|_F.$$

Proof: We define the sets

$$\mathcal{R}(T_{\mathcal{I}\times\mathcal{I}}, \ell, k) := \{X \in \mathbb{R}^{\mathcal{I}\times\mathcal{I}} \mid \operatorname{rank}(X|_{\hat{t}\times\hat{s}}) \leq k \quad \text{for all leaves } t \times s \in \mathcal{L}(T_{\mathcal{I}\times\mathcal{I}}) \text{ or } t \times s \in T_{\mathcal{I}\times\mathcal{I}}^{(\ell)}\}.$$

Obviously M_ℓ is contained in the set $\mathcal{R}(T_{\mathcal{I}\times\mathcal{I}}, \ell, k)$. From one level ℓ to the next level $\ell - 1$, the algorithm determines a best approximation (with respect to the Frobenius norm) of the matrix M_ℓ in the set $\mathcal{R}(T_{\mathcal{I}\times\mathcal{I}}, \ell - 1, k)$:

$$\forall X \in \mathcal{R}(T_{\mathcal{I}\times\mathcal{I}}, \ell - 1, k): \quad \|M_\ell - M_{\ell-1}\|_F \leq \|M_\ell - X\|_F. \tag{6.1}$$

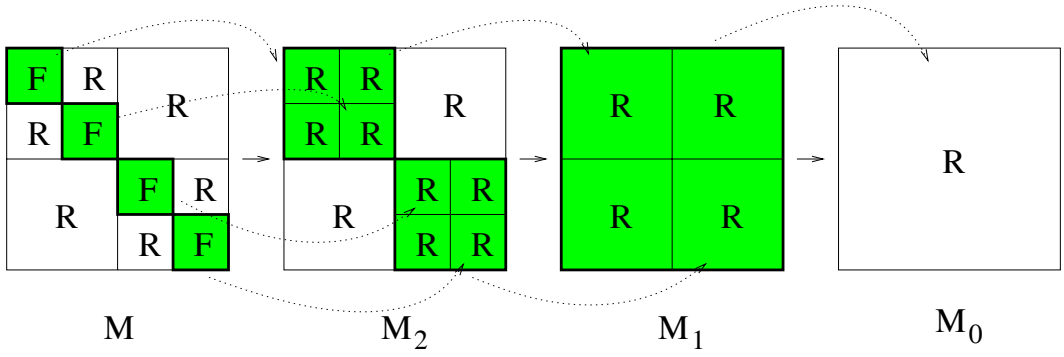


Figure 6.2: The matrix M is converted levelwise for each block: in the first step the `fullmatrix` blocks (F) to `rkmatrix` format (R), then the sons of a block to a single `rkmatrix`.

In the first step (conversion of the `fullmatrix` blocks) this reads

$$\forall X \in \mathcal{R}(T_{\mathcal{I}\times\mathcal{I}}, p, k): \quad \|M - M_p\|_F \leq \|M - X\|_F. \tag{6.2}$$

By induction we prove

$$\|M_\ell - \mathcal{T}_k(M)\|_F \leq 2^{p-\ell}\|M_p - \mathcal{T}_k(M)\|_F$$

as follows. The start $\ell = p$ of the induction is trivial. The induction step $\ell \mapsto \ell - 1$ follows from

$$\|M_{\ell-1} - \mathcal{T}_k(M)\|_F \leq \|M_{\ell-1} - M_\ell\|_F + \|M_\ell - \mathcal{T}_k(M)\|_F \overset{(6.1)}{\leq} 2\|M_\ell - \mathcal{T}_k(M)\|_F.$$

Using this inequality, we can conclude that

$$\begin{aligned}
\|M - M_0\|_F &= \|M - \sum_{\ell=0}^{p-1}(M_\ell - M_{\ell+1}) \; - M_p\|_F \\
&\leq \|M - M_p\|_F + \sum_{\ell=0}^{p-1}\|M_\ell - M_{\ell+1}\|_F \\
&\overset{(6.1),(6.2)}{\leq} \|M - \mathcal{T}_k(M)\|_F + \sum_{\ell=0}^{p-1}\|M_{\ell+1} - \mathcal{T}_k(M)\|_F \\
&\leq \|M - \mathcal{T}_k(M)\|_F + \sum_{\ell=0}^{p-1} 2^{p-\ell-1}\|M_p - \mathcal{T}_k(M)\|_F \\
&\leq 2^p\|M_p - \mathcal{T}_k(M)\|_F + \|M - \mathcal{T}_k(M)\|_F \\
&\leq 2^p(\|M_p - M\|_F + \|M - \mathcal{T}_k(M)\|_F) + \|M - \mathcal{T}_k(M)\|_F \\
&\overset{(6.2)}{\leq} (2^{p+1} + 1)\|M - \mathcal{T}_k(M)\|_F.
\end{aligned}$$

■

The factor '$2^{p+1} + 1 = \mathcal{O}(n)$' in the estimate of the hierarchical approximation error seems to be rather large. Since the singular values in the `rkmatrix` blocks decay rapidly, this factor can easily be compensated without destroying the complexity. In practice the hierarchical approximation error is observed to be much smaller than the estimate.

Next, we want to combine the hierarchical approximation with the multiplication (see Example 6.18). In order to explain the algorithm we will first take a look at a simple example. Let the matrices

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

be given. The goal is to approximate the truncation of the product,

$$C := \mathcal{T}_k(AB) = \mathcal{T}_k\left(\begin{bmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{22}B_{22} + A_{21}B_{12} \end{bmatrix}\right).$$

First, we compute for each block entry the truncated product

$$C'_{ij} := \mathcal{T}_k(A_{i1}B_{1j}), \qquad C''_{ij} := \mathcal{T}_k(A_{i2}B_{2j}).$$

Afterwards we compute the formatted sum

$$C_{ij} := C'_{ij} \oplus C''_{ij}$$

and finally the `rkmatrix` approximation

$$\tilde{C} := \mathcal{T}_k\left(\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}\right).$$

In general, $\tilde{C} \neq C$ due to possible cancellation effects and the hierarchical approximation error.

Implementation 6.23 (Fast Multiplication and Conversion to `rkmatrix` Format) *The matrices* `a` *and* `b` *are given in* `supermatrix` *format (underlying block cluster tree T_a based on T_1, T_2 and T_b based on T_2, T_3) while the target matrix* `r` *is of* `rkmatrix` *format. We compute* `r` $:=$ `r` $\oplus$ `a` $\odot$ `b`.

```
void
addprod2_rkmatrix(prkmatrix r, psupermatrix a, psupermatrix b){
  int i,j,k,bn,bm,bam,no,mo,*rk_no,*rk_mo;
  prkmatrix *rk_r;

  bn = a->block_rows;
  bam = a->block_cols;
  bm = b->block_cols;
  if(a->s!=0x0 && b->s!=0x0){
    rk_no = (int*) malloc(bn*bm*sizeof(int));
    rk_mo = (int*) malloc(bn*bm*sizeof(int));
    rk_r = (prkmatrix*) malloc(bn*bm*sizeof(prkmatrix));
    no = 0;
    for(i=0; i<bn; i++){
      mo = 0;
      for(j=0; j<bm; j++){
        rk_no[i+j*bn] = no;
        rk_mo[i+j*bn] = mo;
        rk_r[i+j*bn] = new_rkmatrix(r->k,a->s[i]->rows,b->s[j*bam]->cols);
        for(k=0; k<bam; k++){
          addprod2_rkmatrix(rk_r[i+j*bn],a->s[i+k*bn],b->s[k+j*bam]);
        }
        mo += b->s[j*bam]->cols;
      }
      no += a->s[i]->rows;
    }
    addparts2_rkmatrix(r,bn*bm,rk_no,rk_mo,rk_r);
    for(i=0; i<bn*bm; i++) del_rkmatrix(rk_r[i]);
    free(rk_r);
    free(rk_mo);
    free(rk_no);
  }else{
```

```
  ...
  /* no hierarchical conversion necessary */
  ...
  }
}
```

In the next chapter we will define the product $T := T_{\mathcal{I}\times\mathcal{I}} \cdot T_{\mathcal{I}\times\mathcal{I}}$ of two block cluster trees. The fast multiplication of two matrices $A, B \in \mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, k)$ is defined in two steps by $C' := \mathcal{T}(A \cdot B)$, where $\mathcal{T}$ is the truncation to the set $\mathcal{H}(T, k)$, and

$$\tilde{C}|_{\hat{t}\times\hat{s}} := \begin{cases} C'|_{\hat{t}\times\hat{s}} & \text{if } t \times s \in \mathcal{L}(T) \\ (C'|_{\hat{t}\times\hat{s}})_{\mathcal{H}} & \text{otherwise,} \end{cases} \qquad \text{for each } t \times s \in \mathcal{L}(T_{\mathcal{I}\times\mathcal{I}}),$$

where $(C'|_{\hat{t}\times\hat{s}})_{\mathcal{H}}$ is the hierarchical approximation of $C'|_{\hat{t}\times\hat{s}}$ from Definition 6.21. The result $\tilde{C}$ belongs to $\mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, k)$, whereas C' belongs to $\mathcal{H}(T, k)$. In Implementation 6.24 we go even one step further: for $t \times s \in T_{\mathcal{I}\times\mathcal{I}}$ the corresponding exact result $C|_{\hat{t}\times\hat{s}}$ is of the form

$$C|_{\hat{t}\times\hat{s}} = \sum_{\ell=1}^{q} A|_{\hat{t}\times\hat{r}_\ell} B|_{\hat{r}_\ell\times\hat{s}}$$

where $r_\ell \in T_{\mathcal{I}}$. Some of the addends are of rank k, but some are hierarchical matrices. We compute hierarchical approximations $(A|_{\hat{t}\times\hat{r}_\ell} B|_{\hat{r}_\ell\times\hat{s}})_{\mathcal{H}}$ of them and afterwards use the fast truncation to sum up all the addends:

$$\begin{aligned} \tilde{C}|^{(1)}_{\hat{t}\times\hat{s}} &:= (A|_{\hat{t}\times\hat{r}_1} B|_{\hat{r}_1\times\hat{s}})_{\mathcal{H}}, \\ \tilde{C}|^{(\ell)}_{\hat{t}\times\hat{s}} &:= \tilde{C}|^{(\ell-1)}_{\hat{t}\times\hat{s}} \oplus (A|_{\hat{t}\times\hat{r}_\ell} B|_{\hat{r}_\ell\times\hat{s}})_{\mathcal{H}}, \quad \ell = 2, \ldots, q, \\ \tilde{C}|_{\hat{t}\times\hat{s}} &:= \tilde{C}|^{(q)}_{\hat{t}\times\hat{s}}. \end{aligned}$$

Implementation 6.24 (Fast Multiplication of Hierarchical Matrices) *The matrices* `a`,`b`,`c` *are given in* `supermatrix` *format (the underlying block cluster tree T_a is based on T_1 and T_2, T_b is based on T_2 and T_3, T_c is based on T_1 and T_3). We compute* $\mathtt{c} := \mathtt{c} \oplus \mathtt{a} \odot \mathtt{b}$.

```
void
muladd_supermatrix(psupermatrix c, psupermatrix a, psupermatrix b){
  int i,j,k,bn,bm,bam;

  bn = c->block_rows;
  bm = c->block_cols;
  bam = a->block_cols;

  if(c->s!=0x0){
    if(a->s!=0x0 && b->s!=0x0){
      /* only supermatrices -> recursion */
      for(i=0; i<bn; i++)
        for(j=0; j<bm; j++)
          for(k=0; k<bam; k++)
            muladd_supermatrix(c->s[i+j*bn],a->s[i+k*bn],b->s[k+j*bam]);
    }else{
      /* a or b is rk or fullmatrix */
      ...
    }
  }else{
    if(c->r!=0x0){
      /* product of 2 supermatrices to be stored in a rkmatrix*/
```

```
        addprod2_rkmatrix(c->r,a,b);
      }else{
        /* c is fullmatrix */
        ...
      }
    }
  }
```

6.2.2 Inversion

Definition 6.25 (Preliminary Formatted Inversion) *Let $T_{\mathcal{I}\times\mathcal{I}}$ be a block cluster tree and $k \in \mathbb{N}$. The preliminary formatted inversion operator is defined as*

$$\widetilde{\mathrm{Inv}} : \{M \in \mathbb{R}^{n\times n} \mid \mathrm{rank}(M) = n\} \to \mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, k), \qquad M \mapsto \mathcal{T}_k(M^{-1}).$$

Since the (`fullmatrix`) inversion of M is rather expensive, we will present an algorithm that computes an approximation to $\mathcal{T}_k(M^{-1})$ without the need to invert M exactly. This approximation will not necessarily be a best approximation.

The inversion is done by use of the equation

$$M^{-1} = \begin{bmatrix} M_{11}^{-1} + M_{11}^{-1}M_{12}S^{-1}M_{21}M_{11}^{-1} & -M_{11}^{-1}M_{12}S^{-1} \\ -S^{-1}M_{21}M_{11}^{-1} & S^{-1} \end{bmatrix}, \qquad M = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix}, \tag{6.3}$$

where $S = M_{22} - M_{21}M_{11}^{-1}M_{12}$. If the inversion of the submatrices M_{11} and S is already done, then we only need to perform multiplications and additions of submatrices. These can be replaced by the formatted operations $\oplus$ and $\odot$. Recursively, we get an approximate inverse $\mathrm{Inv}_{\mathcal{H}}(M)$. This approximation is called the formatted inverse.

Our goal is to derive an algorithm that computes the inverse of M with a minimum of storage overhead. Later we will see that it is possible to compute the approximate inverse within the same `supermatrix` structure as the input matrix M, overwriting the input data by the output data, essentially without any storage overhead. Algorithm 5 computes the approximate inverse in `supermatrix` format and uses a second `supermatrix` as temporary storage, destroying the content of the input matrix and overwriting it by the approximate inverse.

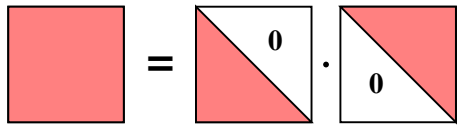
Algorithm 5 The $\mathcal{H}$-matrix inversion $\mathrm{Inv}_{\mathcal{H}}(M)$ for 2×2 block structures. On input `m` is the matrix to be inverted, on output `m` is the approximate inverse $\mathrm{Inv}_{\mathcal{H}}(M)$. `x` is used for temporary storage.

procedure HInvert(**var** `m`, **var** `x`)
if the matrix `m` is of the structure (6.3) **then**
  HInvert($\mathtt{m}_{11}$, $\mathtt{x}_{11}$) { destroys $\mathtt{m}_{11}$ }
  $\mathtt{x}_{12} := \mathtt{x}_{11} \odot \mathtt{m}_{12}$ { $\mathtt{x}_{12} = M_{11}^{-1}M_{12}$ }
  $\mathtt{x}_{21} := \mathtt{m}_{21} \odot \mathtt{x}_{11}$ { $\mathtt{x}_{21} = M_{21}M_{11}^{-1}$ }
  $\mathtt{m}_{22} := \mathtt{m}_{22} \ominus \mathtt{m}_{21} \odot \mathtt{x}_{12}$ { $\mathtt{m}_{22} = S$ }
  HInvert($\mathtt{m}_{22}$, $\mathtt{x}_{22}$) { $\mathtt{m}_{22} = S^{-1}$, destroys $\mathtt{x}_{22}$ }
  $\mathtt{m}_{12} := -\mathtt{x}_{12} \odot \mathtt{m}_{22}$ { $\mathtt{m}_{12} = -M_{11}^{-1}M_{12}S^{-1}$ }
  $\mathtt{m}_{11} := \mathtt{m}_{11} \ominus \mathtt{m}_{12} \odot \mathtt{x}_{21}$ { $\mathtt{m}_{11} = M_{11}^{-1} + M_{11}^{-1}M_{12}S^{-1}M_{21}M_{11}^{-1}$ }
  $\mathtt{m}_{21} := -\mathtt{m}_{22} \odot \mathtt{x}_{21}$ { $\mathtt{m}_{21} = -S^{-1}M_{21}M_{11}^{-1}$ }
else
  Invert($\mathtt{m}_{11}$) { `fullmatrix` inversion }
end if

6.2.3 Cholesky and LU Decomposition

Sometimes one does not need the whole (approximate) inverse but only a method to perform the matrix vector multiplication $b \mapsto A^{-1}b$, i.e., to solve the system $Ax = b$. In that case it is sufficient to compute a Cholesky or LU decomposition

$$A \approx LU$$

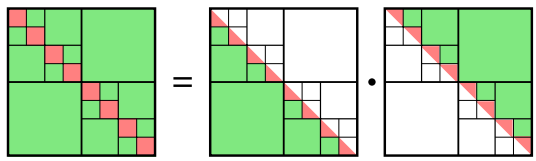


of the matrix A. Depending on the application one could want a decomposition such that $\|A - LU\|$ is of the size of the discretisation error or just a coarse approximation in order to precondition the linear system and use a simple iterative solver, e.g., GMRES:

$$(LU)^{-1}Ax = (LU)^{-1}b$$

An (approximate) $\mathcal{H}$-LU decomposition is defined as a decomposition of the form

$$A \approx L_{\mathcal{H}}U_{\mathcal{H}}$$



where the two (lower and upper triangular) matrices $L_{\mathcal{H}}$ and $U_{\mathcal{H}}$ are stored in the $\mathcal{H}$-matrix format. As for the `fullmatrix` version one can store the two factors by overwriting the given $\mathcal{H}$-matrix A.

Three procedures are needed to compute the decomposition: first, a method to solve a triangular $\mathcal{H}$-matrix system for a vector. Second, a method to solve a triangular $\mathcal{H}$-matrix system for a matrix and third the LU-decomposition which is based on the aforementioned two.

Solving a triangular system $Lx = b$ for a right-hand side b and lower triangular matrix L in the $\mathcal{H}$-matrix format is done recursively:

- if L is not subdivided (`fullmatrix`) then use the LAPACK subroutine `dtrsv`.
- if L is subdivided, for simplicity into `block_rows`=2 times `block_cols`=2 submatrices

$$L = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix}, x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, b = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

  then we have to solve

$$L_{11}x_1 = b_1$$

  which yields x_1 and afterwards

$$L_{22}x_2 = b_1 - L_{21}x_1$$

  which yields x_2.

Solving a triangular system $LX = B$ for a right-hand side matrix B and lower triangular matrix L in the $\mathcal{H}$-matrix format is done similarly by recursion:

- if L is not subdivided (`fullmatrix`) then compute the solution row-wise by the LAPACK subroutine `dtrsv`.
- if L is subdivided, for simplicity into `block_rows`=2 times `block_cols`=2 submatrices

$$L = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix}, X = \begin{bmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{bmatrix}, B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

then we have to solve

$$L_{11}X_{11} = B_{11}, \quad L_{11}X_{12} = B_{12}$$

which yields X_{11}, X_{12} and afterwards

$$L_{22}X_{21} = B_{21} - L_{21}X_{11}, \quad L_{22}X_{22} = B_{22} - L_{21}X_{12}$$

which yields X_{21}, X_{22}.

Analogously we solve an upper triangular system $XU = B$.

Finally, the $\mathcal{H}$-LU decomposition $A = LU$ is defined recursively by

- if A is not subdivided (`fullmatrix`) then use the LAPACK subroutine `dgetrf`.
- if A is subdivided, for simplicity into `block_rows`=2 times `block_cols`=2 submatrices

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

then we have to compute the factorisation

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

which yields the upper left components L_{11}, U_{11} of the factors L, U, solve the triangular systems

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

and compute again a factorisation

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

For symmetric matrices A one can of course compute a Cholesky or LDL^T factorisation with half the complexity, where the steps in the algorithm are just analogously to those for the LU decomposition.

The LU decomposition of a matrix M uses only the input matrix as storage and overwrites it by the factors L and U. These factors can then be used to compute the approximate inverse $\mathrm{Inv}_{\mathcal{H}}(M)$ by overwriting M, i.e., the factors L and U. In order to do this efficiently, we use the following special case of a multiplication.

Let L be a square lower trianguar and X a general 2×2 block matrix of matching size so that the multiplication $L \cdot X$ can be formulated blockwise,

$$L \cdot X = \begin{bmatrix} L_{11}X_{11} & L_{11}X_{12} \\ L_{21}X_{11} + L_{22}X_{21} & L_{21}X_{12} + L_{22}X_{22} \end{bmatrix}. \tag{6.4}$$

Then the product can be computed in the order presented in Algorithm 6, overwriting X by the product LX without using extra storage. The multiplication with an upper triangular matrix from the right can be done analogously.

The inverse of a lower triangular matrix L of 2×2 block structure,

$$L = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix}, \quad L^{-1} = \begin{bmatrix} L_{11}^{-1} & 0 \\ -L_{22}^{-1}L_{21}L_{11}^{-1} & L_{22}^{-1} \end{bmatrix}, , \tag{6.5}$$

can also be computed by overwriting the input matrix with the (approximate) inverse, cf. Algorithm 7.

Now we can formulate the matrix inversion via LU decompositions so that only a single `supermatrix` is necessary for the whole inversion process, namely the input matrix which is overwritten by the approximate) inverse in Algorithm 8.

Algorithm 6 The $\mathcal{H}$-matrix multiplication $X := L \cdot X$ for 2×2 block structures, where L is lower triangular. On input $\mathtt{X}$ is the matrix to be multiplied, on output $\mathtt{X}$ is the product.

procedure MultiplyLowerLeft($\mathtt{L}$, **var** $\mathtt{X}$)
if L and X are subdivided as in (6.4) **then**
  MultiplyLowerLeft($\mathtt{L}_{22}$, $\mathtt{X}_{21}$)
  MultiplyLowerLeft($\mathtt{L}_{22}$, $\mathtt{X}_{22}$)
  $\mathtt{X}_{21} := \mathtt{X}_{21} \oplus \mathtt{L}_{21} \odot \mathtt{X}_{11}$
  $\mathtt{X}_{22} := \mathtt{X}_{22} \oplus \mathtt{L}_{21} \odot \mathtt{X}_{12}$
  MultiplyLowerLeft($\mathtt{L}_{11}$, $\mathtt{X}_{11}$)
  MultiplyLowerLeft($\mathtt{L}_{11}$, $\mathtt{X}_{12}$)
else
  $\{\mathtt{L} = (\mathtt{L}_{i,j})_{i,j\in\{1,\dots,n\}}, \mathtt{X} = (\mathtt{X}_{i,j})_{i\in\{1,\dots,n\},j\in\{1,\dots,m\}}\}$
  for $i = n, \dots, 1$ **do**
    for $j = 1, \dots, m$ **do**
      $\mathtt{X}_{i,j} := \mathtt{L}_{i,i}\mathtt{X}_{i,j}$
    end for
    for $\ell = i-1, \dots, 1$ **do**
      for $j = 1, \dots, m$ **do**
        $\mathtt{X}_{i,j} := \mathtt{X}_{i,j} + \mathtt{L}_{i,\ell}\mathtt{X}_{\ell,j}$ { uses only the unmodified $\mathtt{X}_{\ell,j}$ for $\ell < i$ }
      end for
    end for
  end for
end if

Algorithm 7 The $\mathcal{H}$-matrix inversion $L := L^{-1}$ for 2×2 block structures. On input $\mathtt{L}$ is the lower triangular matrix to be inverted, on output $\mathtt{L}$ is the approximate inverse.

procedure InvertLowerTriangular($\mathtt{L}$)
if L is subdivided as in (6.5) **then**
  InvertLowerTriangular($\mathtt{L}_{11}$) { $\mathtt{L}_{11} = L_{11}^{-1}$ }
  InvertLowerTriangular($\mathtt{L}_{22}$) { $\mathtt{L}_{22} = L_{11}^{-1}$ }
  MultiplyLowerLeft($\mathtt{L}_{22}$, $\mathtt{L}_{21}$) { $\mathtt{L}_{21} = L_{22}^{-1}L_{21}$ }
  MultiplyLowerRight($-\mathtt{L}_{11}$, $\mathtt{L}_{21}$) { $\mathtt{L}_{21} = -L_{22}^{-1}L_{21}L_{11}^{-1}$ }
else
  $\{\mathtt{L} = (\mathtt{L}_{i,j})_{i,j\in\{1,\dots,n\}}\}$
  for $i = 1, \dots, n$ **do**
    $L_{i,i} := L_{i,i}^{-1}$
  end for
  for $i = n, \dots, 1$ **do**
    for $j = i, \dots, 1$ **do**
      $\mathtt{L}_{i,j} := \mathtt{L}_{i,i}\mathtt{L}_{i,j}$
      for $\ell = j+1, \dots, i-1$ **do**
        $\mathtt{L}_{i,j} := \mathtt{L}_{i,j} + \mathtt{L}_{i,\ell}\mathtt{L}_{\ell,j}$
        { $\mathtt{L}_{i,\ell}$ is a block of the inverse, because $\ell > j$, and $\mathtt{L}_{\ell,j}$ is unmodified because $\ell < i$ }
      end for
      $\mathtt{L}_{i,j} := -\mathtt{L}_{i,j}\mathtt{L}_{j,j}$
    end for
  end for
end if

Algorithm 8 The $\mathcal{H}$-matrix inversion $A := A^{-1}$ for 2×2 block structures. On input $\mathtt{A}$ is the matrix to be inverted, on output $\mathtt{A}$ is the approximate inverse.

procedure Invert($\mathtt{M}$)
LUDecomposition($\mathtt{M}$) { diag(U) = 1, L stored in lower triangular part of M, }
{ strictly upper triangular part of U stored in strictly upper triangular part of M }
InvertLowerTriangular($\mathtt{M}$)
InvertUpperTriangular($\mathtt{M}$)
MultiplyUL($\mathtt{M}$)

procedure MultiplyUL($\mathtt{M}$)
if M is subdivided as in (6.3) **then**
MultiplyUL($\mathtt{M}_{11}$)
$\mathtt{M}_{11} := M_{11} \oplus \mathtt{M}_{12} \odot \mathtt{M}_{21}$ { $\mathtt{M}_{11} = U_{11}L_{11} + U_{12}L_{21}$ }
MultiplyLowerRight($\mathtt{M}_{22}$,$\mathtt{M}_{12}$) { $\mathtt{M}_{12} = U_{12}L_{22}$ }
MultiplyUpperLeft($\mathtt{M}_{22}$,$\mathtt{M}_{21}$) { $\mathtt{M}_{21} = U_{22}L_{21}$ }
MultiplyUL($\mathtt{M}_{22}$)
else
{ $\mathtt{M} = (\mathtt{M}_{i,j})_{i,j\in\{1,\ldots,n\}}$ }
for $i = 1, \ldots, n$ **do**
for $j = 1, \ldots, i-1$ **do**
$\mathtt{M}_{j,i} := \mathtt{M}_{j,i}\mathtt{M}_{i,i}$ { $\mathtt{M}_{j,i} = U_{j,i}L_{i,i}$, $\mathtt{M}_{i,j} = U_{i,i}L_{i,j} = L_{i,j}$ }
for $\ell = i+1, \ldots, n$ **do**
$\mathtt{M}_{i,j} := \mathtt{M}_{i,j} + \mathtt{M}_{i,\ell}\mathtt{M}_{\ell,j}$ { $\mathtt{M}_{i,j} := \mathtt{M}_{i,j} + U_{i,\ell}L_{\ell,j}$ }
$\mathtt{M}_{j,i} := \mathtt{M}_{j,i} + \mathtt{M}_{j,\ell}\mathtt{M}_{\ell,i}$ { $\mathtt{M}_{j,i} := \mathtt{M}_{j,i} + U_{j,\ell}L_{\ell,i}$ }
end for
end for
for $\ell = i+1, \ldots, n$ **do**
$\mathtt{M}_{i,i} := \mathtt{M}_{i,i} + \mathtt{M}_{i,\ell}\mathtt{M}_{\ell,i}$ { $\mathtt{M}_{i,i} := \mathtt{M}_{i,i} + U_{i,\ell}L_{\ell,i}$ }
end for
end for
end if

Chapter 7

Complexity Estimates

The complexity estimates for the arithmetics of hierarchical matrices can be decomposed into two parts:

a) The storage, matrix-vector multiplication and addition require the so-called *sparsity* of the underlying block cluster tree $T_{\mathcal{I}\times\mathcal{I}}$.

b) For the (formatted) multiplication and inversion in the $\mathcal{H}$-matrix format we need the so-called *idempotency* of $T_{\mathcal{I}\times\mathcal{I}}$.

The estimates in this general form are contained in [27] (german) and [29] (english). For the one-dimensional case the complexity estimates can be simplified as in [40], and for a two- and three-dimensional model problem the storage, matrix-vector mutliplication and addition can be estimated as in [42].

7.1 Arithmetics in the `rkmatrix` Representation

Let $M = AB^T \in \mathbb{R}^{n\times m}$ be a matrix in `rkmatrix` representation. Since only the two factors A, B need to be stored, the storage requirements amount to

$$N_{St,R}(k,n,m) = k(n+m) \tag{7.1}$$

while in the `fullmatrix` representation we have

$$N_{St,F}(n,m) = nm. \tag{7.2}$$

7.1.1 Reduced Singular Value Decomposition (rSVD)

Let $M = AB^T \in \mathbb{R}^{n\times m}$ be a matrix in `rkmatrix` representation. We computed an rSVD $M = U\Sigma V^T$ by

1. Computing (reduced) QR-factorisations of A, B: $A = Q_A R_A$, $B = Q_B R_B$ with matrices $Q_A \in \mathbb{R}^{n\times k}$, $Q_B \in \mathbb{R}^{m\times k}$, $R_A, R_B \in \mathbb{R}^{k\times k}$.
2. Computing an rSVD of $R_A R_B^T = U'\Sigma V'$.
3. Computing $U := Q_A U'$, $V := Q_B V'$.

The complexity of each step is

QR-factorisation of A	$\mathcal{O}(nk^2)$
QR-factorisation of B	$\mathcal{O}(mk^2)$
Multiplication $R_A R_B^T$	$\mathcal{O}(k^3)$
rSVD of $R_A R_B^T$	$\mathcal{O}(k^3)$
Multiplication $U := Q_A U'$	$\mathcal{O}(nk^2)$
Multiplication $V := Q_B V'$	$\mathcal{O}(mk^2)$
Altogether	$\mathcal{O}((n+m)k^2)$

Lemma 7.1 (Truncation) *The truncation $\mathcal{T}_k$ of a matrix $M \in \mathbb{R}^{n\times m}$ in* `rkmatrix` *format to lower rank $k' < k$ is of complexity*

$$N_{\mathcal{T}}(k,n,m) \le 6k^2(n+m) + 23k^3.$$

Remark 7.2 (Large k) *If $k > \min(n,m)$ then the computation of the rSVD of $R_A R_B^T$ can be exceedingly expensive. In order to avoid this, we first compare k, n, m and if $k > \min(n,m)$ then we first change the representation of M to* `fullmatrix` *by $M_{ij} := \sum_{\nu=1}^{k} A_{i\nu}B_{j\nu}$ at a cost of knm and afterwards we compute an rSVD of M in* `fullmatrix` *representation in $\mathcal{O}(\min(n,m)^2 \max(n,m))$. Altogether this amounts to $\mathcal{O}(\min(n,m)\max(n,m)k)$.*

7.1.2 Formatted `rkmatrix` Arithmetics

Multiplication

The multiplication of an `rkmatrix` $R = AB^T = \sum_{\nu=1}^{k} A_\nu B_\nu^T$ with a matrix M involves k times the matrix-vector multiplication of the matrix M or M^T:

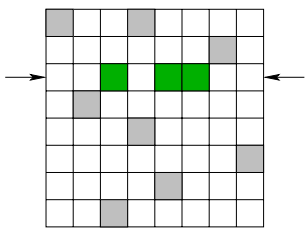
$$\begin{aligned} RM &= AB^TM = \sum_{\nu=1}^{k} A_\nu (M^T B_\nu)^T, \\ MR &= MAB^T = \sum_{\nu=1}^{k} (MA_\nu) B_\nu^T. \end{aligned}$$

Addition

The formatted addition $A \oplus B := \mathcal{T}_k(A+B)$ of two matrices in `rkmatrix` format is done by truncation of the exact sum (rank $2k$) with a complexity of $\mathcal{O}((n+m)k^2)$.

7.2 Arithmetics in the $\mathcal{H}$-Matrix Format

For a matrix $M \in \mathbb{R}^{\mathcal{I}\times\mathcal{I}}$ one can count the number of nonzero entries per row,

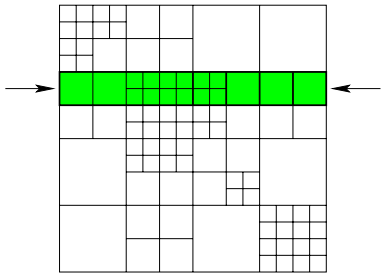


$$c := \max_{i\in\mathcal{I}} \#\{j \in \mathcal{I} \mid M_{ij} \neq 0\},$$

such that the number of nonzero entries in the whole matrix is at most $c\#\mathcal{I}$. The constant c depends on the sparsity pattern and for standard FEM stiffness matrices the constant c is independent of the size of $\#\mathcal{I}$.

The block cluster tree $T_{\mathcal{I}\times\mathcal{I}}$ may have a similar sparsity property which is measured by the quantity C_{sp} defined below. In Section 7.3, the construction of $T_{\mathcal{I}}$ and $T_{\mathcal{I}\times\mathcal{I}}$ will lead to a block cluster tree with a sparsity constant C_{sp} independent of the size of $\#\mathcal{I}$.

Definition 7.3 (Sparsity) *Let $T_{\mathcal{I}\times\mathcal{I}}$ be a block cluster tree based on $T_{\mathcal{I}}$. We define the* sparsity (constant) *C_{sp} of $T_{\mathcal{I}\times\mathcal{I}}$ by*



$$C_{\mathrm{sp}} := \max\Big\{ \begin{array}{l} \max_{r\in T_{\mathcal{I}}} \#\{s \in T_{\mathcal{I}} \mid r \times s \in T_{\mathcal{I}\times\mathcal{I}}\}, \\ \max_{s\in T_{\mathcal{I}}} \#\{r \in T_{\mathcal{I}} \mid r \times s \in T_{\mathcal{I}\times\mathcal{I}}\} \end{array} \Big\}.$$

Lemma 7.4 *For any block cluster tree $T_{\mathcal{I}\times\mathcal{I}}$ based on $T_{\mathcal{I}}$ with sparsity constant C_{sp} and depth p holds*

$$\sum_{t\times s\in T_{\mathcal{I}\times\mathcal{I}}} \#\hat{t} + \#\hat{s} \leq 2C_{\mathrm{sp}}(p+1)\#\mathcal{I}.$$

Proof:

$$\begin{aligned}
\sum_{t\times s\in T_{\mathcal{I}\times\mathcal{I}}} \#\hat{t} + \#\hat{s} &= \sum_{i=0}^{p} \sum_{t\times s\in T^{(i)}_{\mathcal{I}\times\mathcal{I}}} \#\hat{t} + \#\hat{s} \\
&= \sum_{i=0}^{p} \sum_{t\times s\in T^{(i)}_{\mathcal{I}\times\mathcal{I}}} \#\hat{t} + \sum_{i=0}^{p} \sum_{t\times s\in T^{(i)}_{\mathcal{I}\times\mathcal{I}}} \#\hat{s} \\
&\leq 2\sum_{i=0}^{p} \sum_{t\in T^{(i)}_{\mathcal{I}}} C_{\mathrm{sp}}\#\hat{t} \\
&\overset{\text{Lemma 6.20}}{\leq} 2\sum_{i=0}^{p} C_{\mathrm{sp}}\#\mathcal{I} = 2C_{\mathrm{sp}}(p+1)\#\mathcal{I}.
\end{aligned}$$

■

Definition 7.5 (Admissible and Inadmissible Leaves) *Let $T := T_{\mathcal{I}\times\mathcal{I}}$ be a block cluster tree. The set of leaves of T is denoted by $\mathcal{L}(T)$ We define the set of admissible leaves of T as*

$$\mathcal{L}^+(T) := \{t \times s \in T \mid t \times s \text{ admissible}\}$$

and the set of inadmissible leaves of T as

$$\mathcal{L}^-(T) := \{t \times s \in T \mid t \times s \text{ inadmissible}\}.$$

Lemma 7.6 (Storage) *Let $T := T_{\mathcal{I}\times\mathcal{I}}$ be a block cluster tree with sparsity constant C_{sp} and depth p. Let $k \in \mathbb{N}$. The storage requirements $N_{\mathrm{St}}(T,k)$ for a matrix $M \in \mathcal{H}(T,k)$ are bounded by*

$$N_{\mathrm{St}}(T,k) \leq 2C_{\mathrm{sp}}(p+1)\max\{k, n_{\min}\}\#\mathcal{I}.$$

Proof:

$$\begin{aligned}
N_{\mathrm{St}}(T,k) &\overset{(7.1),(7.2)}{=} \sum_{t\times s\in\mathcal{L}^+(T)} k(\#\hat{t} + \#\hat{s}) + \sum_{t\times s\in\mathcal{L}^-(T)} \#\hat{t}\cdot\#\hat{s} \\
&\leq \sum_{t\times s\in\mathcal{L}^+(T)} k(\#\hat{t} + \#\hat{s}) + \sum_{t\times s\in\mathcal{L}^-(T)} n_{\min}(\#\hat{t} + \#\hat{s}) \\
&\leq \sum_{t\times s\in T} \max\{n_{\min}, k\}(\#\hat{t} + \#\hat{s}) \\
&\overset{L.7.4}{\leq} 2C_{\mathrm{sp}}\max\{k, n_{\min}\}(p+1)\#\mathcal{I}.
\end{aligned}$$

■

Lemma 7.7 (Matrix-Vector Multiplication) *Let $T := T_{\mathcal{I}\times\mathcal{I}}$ be a block cluster tree. Let $k \in \mathbb{N}$. The complexity $N_{\mathcal{H}\cdot v}(T,k)$ of the matrix-vector multiplication for a matrix $M \in \mathcal{H}(T,k)$ are bounded by*

$$N_{\mathcal{H}\cdot v}(T,k) \leq 2N_{\mathrm{St}}(T,k).$$

Proof: Consider the matrix-vector product blockwise and use (7.1), (7.2) together with the respective counterparts for the matrix-vector product. ∎

7.2.1 Truncation

Lemma 7.8 (Cardinality of $T_{\mathcal{I}}$ and $T_{\mathcal{I}\times\mathcal{I}}$) *Let $T_{\mathcal{I}}$ be a cluster tree of depth $p \geq 1$ and $T := T_{\mathcal{I}\times\mathcal{I}}$ a block cluster tree with sparsity constant C_{sp}.*

1. *If $\#\operatorname{sons}(t) \neq 1$ holds for all (or at least $\#\mathcal{I}/p$) nodes $t \in T_{\mathcal{I}}$ then*

$$\#T_{\mathcal{I}} \;\leq\; 2\#\mathcal{I}, \qquad \#T \;\leq\; 2C_{\mathrm{sp}}\#\mathcal{I}. \tag{7.3}$$

2. *If $\#\operatorname{sons}(t) \neq 1$ is not necessarily fulfilled then*

$$\#T_{\mathcal{I}} \;\leq\; 2p\#\mathcal{I}, \qquad \#T \;\leq\; 2pC_{\mathrm{sp}}\#\mathcal{I}. \tag{7.4}$$

Proof: Exercise. ∎

Lemma 7.9 (Truncation $\mathcal{T}_k$) *Let $T := T_{\mathcal{I}\times\mathcal{I}}$ be a block cluster tree. The truncation $\mathcal{T}_k$ for a matrix $M \in \mathcal{H}(T,k)$ in* `supermatrix` *format to lower rank $k' < k$ is of complexity*

$$N_{\mathcal{T}}(T,k) \leq 6kN_{\mathrm{St}}(T,k) + 23k^3\#\mathcal{L}(T).$$

Proof:

$$\begin{aligned} N_{\mathcal{T}}(T,k) &= \sum_{t\times s\in\mathcal{L}^+(T)} N_{\mathcal{T}}(k,\#\hat{t},\#\hat{s}) \overset{\text{Lemma 7.1}}{\leq} \sum_{t\times s\in\mathcal{L}^+(T)} 6k^2(\#\hat{t}+\#\hat{s}) + 23k^3\#\mathcal{L}(T) \\ &\leq 6kN_{\mathrm{St}}(T,k) + 23k^3\#\mathcal{L}(T). \end{aligned}$$

∎

Lemma 7.10 (Fast Truncation $\mathcal{T}'_k$) *Let $T := T_{\mathcal{I}\times\mathcal{I}}$ be a block cluster tree. The truncation $\mathcal{T}'_k$ for a matrix $M \in \mathcal{H}(T,qk)$ in* `supermatrix` *format to rank k is of complexity*

$$N_{\mathcal{T}'}(T,qk) \leq 24(q-1)kN_{\mathrm{St}}(T,k) + 184(q-1)k^3\#\mathcal{L}(T).$$

Proof: Apply $(q-1)$-times Lemma 7.9 for rank $2k$. ∎

Lemma 7.11 (Hierarchical Approximation) *Let $T := T_{\mathcal{I}\times\mathcal{I}}$ be a block cluster tree of depth p based on the cluster tree $T_{\mathcal{I}}$ where each node has at most C_{sons} sons (typically $C_{\mathrm{sons}} = 2$) and $n_{min} \leq k$ (for simplification). Then the complexity to compute the hierarchical approximation (cf. Definition 6.21) is bounded by*

$$N_{\mathcal{H}apx}(T,k) \;\leq\; 12C_{\mathrm{sp}}\max\{4, C_{\mathrm{sons}}^2\}(p+1)k^2\#\mathcal{I} \;+\; 23C_{\mathrm{sons}}^3k^3\#T.$$

Proof: First step of the algorithm: the `fullmatrix` blocks corresponding to leaves $t \times s$ of T have to be truncated to `rkmatrix` format. Since either $\#\hat{t} \leq n_{\min}$ or $\#\hat{s} \leq n_{\min}$ the complexity for the rSVD is at most $21(\#\hat{s} + \#\hat{t})n_{min}^2$. In the later steps we always have to truncate a $\hat{t} \times \hat{s}$ `supermatrix` consisting of submatrices in `rkmatrix` format to `rkmatrix` format. The submatrices can be extended by zeros to yield an

$\hat{t} \times \hat{s}$ `rkmatrix` of rank at most $C_{\text{sons}}^2 k$. The truncation to rank k is according to Lemma 7.1 of complexity $6C_{\text{sons}}^2 k^2(\#\hat{s} + \#\hat{t}) + 23C_{\text{sons}}^3 k^3$. For all nodes this sums up to

$$\begin{aligned} N_{\mathcal{H}apx}(T,k) &\leq \sum_{t\times s\in\mathcal{L}^-(T)} 21 n_{min}^2(\#\hat{s}+\#\hat{t}) + \sum_{t\times s\in T\setminus\mathcal{L}^-(T)} \left(6C_{\text{sons}}^2 k^2(\#\hat{s}+\#\hat{t}) + 23C_{\text{sons}}^3 k^3\right) \\ &\leq \sum_{t\times s\in T} \left(\max\{21, 6C_{\text{sons}}^2\} k^2(\#\hat{s}+\#\hat{t}) + 23C_{\text{sons}}^3 k^3\right) \\ &\overset{L.7.4}{\leq} 2C_{\text{sp}}(p+1)\left(\max\{21, 6C_{\text{sons}}^2\} k^2 \#\mathcal{I}\right) + 23C_{\text{sons}}^3 k^3 \#T. \end{aligned}$$

■

7.2.2 Addition

The complexity estimate for the formatted addition of two $\mathcal{H}$-matrices based on the same block cluster tree $T_{\mathcal{I}\times\mathcal{I}}$ can be derived directly from the estimate for the truncation.

Lemma 7.12 (Addition) *Let $T = T_{\mathcal{I}\times\mathcal{I}}$ be a block cluster tree and $k \geq 1$. Then the complexity of the formatted addition of two matrices from $\mathcal{H}(T,k)$ is bounded by*

$$N_{\oplus}(T,k) \leq 24kN_{\text{St}}(T,k) + 184k^3 \#\mathcal{L}(T).$$

Proof:

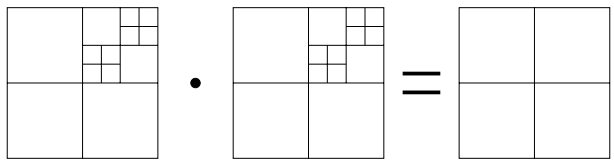
$$\begin{aligned} N_{\oplus}(T,k) &= \sum_{t\times s\in\mathcal{L}^+(T)} N_{\mathcal{T}}(2k, \#\hat{t}, \#\hat{s}) + \sum_{t\times s\in\mathcal{L}^-(T)} \#\hat{t}\#\hat{s} \\ &\overset{\text{Lemma 7.1}}{\leq} \sum_{t\times s\in\mathcal{L}^+(T)} 24k^2(\#\hat{t}+\#\hat{s}) + 184k^3\#\mathcal{L}^+(T) + \sum_{t\times s\in\mathcal{L}^-(T)} \#\hat{t}\#\hat{s} \\ &\leq 24kN_{\text{St}}(T,k) + 184k^3\#\mathcal{L}(T). \end{aligned}$$

■

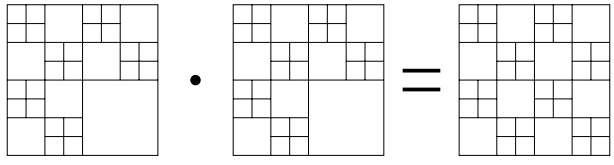
7.2.3 Multiplication

The multiplication is a much more complicated operation than the addition, as we have already seen in Examples 6.16, 6.18. In order to illustrate what might happen during the multiplication we will take a look at three typical examples, where we multiply two matrices $A, B \in \mathcal{H}(T,k)$ and seek a block cluster tree $T \cdot T$ (the product tree) in such a way that $A \cdot B \in \mathcal{H}(T \cdot T, \tilde{k})$.

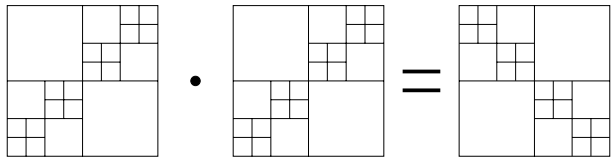
It may happen that the structure of the product of two matrices from $\mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, k)$ is "coarser" than $T_{\mathcal{I}\times\mathcal{I}}$:



The coarsening effect is not severe, since the structure can be refined to fit to the original tree $T := T_{\mathcal{I}\times\mathcal{I}}$, but it may also happen that the structure of the product of two matrices from $\mathcal{H}(T,k)$ is "finer" than T:



The refinement effect seems to suggest that the structure of the product of two matrices from $\mathcal{H}(T,k)$ is just slightly enriched but it may also change totally:

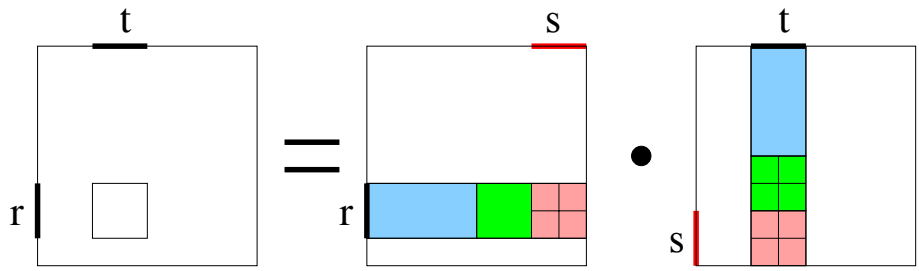


This poses the question how one can efficiently describe the structure of the product. The product tree $T \cdot T$ should be a block cluster tree based on the cluster tree $T_{\mathcal{I}}$, i.e.,

$$\operatorname{root}(T \cdot T) := \mathcal{I} \times \mathcal{I}.$$

If $r \times t$ is a node in the product tree $T \cdot T$ then

$$(AB)|_{\hat{r}\times\hat{t}} = A|_{\hat{r}\times\mathcal{I}} \cdot B|_{\mathcal{I}\times\hat{t}}.$$



If there is a cluster $s \in T_{\mathcal{I}}$ such that both $r \times s$ and $s \times t$ are a non-leaf element of T, then

$$(AB)|_{\hat{r}\times\hat{t}} = A|_{\hat{r}\times\mathcal{I}\setminus\hat{s}} \cdot B|_{\mathcal{I}\setminus\hat{s}\times\hat{t}} + A|_{\hat{r}\times\hat{s}} \cdot B|_{\hat{s}\times\hat{t}}.$$

The product $A|_{\hat{r}\times\hat{s}} \cdot B|_{\hat{s}\times\hat{t}}$ of the two subdivided matrices will yield a matrix that is again subdivided, therefore the node $r \times t$ in the product tree must not be a leaf. This leads to the definition

Definition 7.13 (Product Tree) *We define the product $T \cdot T$ of a block cluster tree T based on the cluster tree $T_{\mathcal{I}}$ by the root* $\operatorname{root}(T \cdot T) := \mathcal{I} \times \mathcal{I}$ *and for each node $r \times t$ of the product tree the successors*

$$\operatorname{sons}(r \times t) := \left\{ r' \times t' \;\middle|\; \exists s, s' \in T_{\mathcal{I}} :\; r' \times s' \in \operatorname{sons}_T(r \times s), s' \times t' \in \operatorname{sons}_T(s \times t) \right\}.$$

The sparsity of the block cluster tree T carries over to the product tree $T \cdot T$.

Lemma 7.14 *The product tree $T \cdot T$ is a block cluster tree of depth at most* $\operatorname{depth}(T)$. *If $C_{\mathrm{sp}}(T)$ is the sparsity constant of T then the sparsity of the product tree is bounded by the product of the sparsity:*

$$C_{\mathrm{sp}}(T \cdot T) \leq C_{\mathrm{sp}}(T)^2.$$

Proof: Due to the symmetry of the sparsity we only give a rowwise bound. Let $r \in T_{\mathcal{I}}$. Then

$$\begin{aligned} \{t \in T_{\mathcal{I}} \mid r \times t \in T \cdot T\} &\subset \{t \in T_{\mathcal{I}} \mid \exists s \in T_{\mathcal{I}} : r \times s \in T, s \times t \in T\}, \\ \#\{t \in T_{\mathcal{I}} \mid r \times t \in T \cdot T\} &\leq \sum_{s \in T_{\mathcal{I}}, r \times s \in T} \#\{t \in T_{\mathcal{I}} \mid s \times t \in T\} \\ &\leq C_{\mathrm{sp}}(T) C_{\mathrm{sp}}(T). \end{aligned}$$

■

The product tree $T \cdot T$ allows us to describe the result of the multiplication blockwise in a compact form. In order to simplify the notation we need to define the ancestors of a cluster.

Definition 7.15 (Ancestors) *Let $T_{\mathcal{I}}$ be a cluster tree and $t \in T_{\mathcal{I}}^{(\ell)}$. Then we define the ancestor of t on level $j \leq \ell$ as the uniquely determined vertex $\mathcal{F}^j(t) \in T_{\mathcal{I}}^{(j)}$ with $\hat{t} \subset \widehat{\mathcal{F}^j(t)}$. If t is an ancestor of s, we also write $s \in S^*(t)$, i.e., $\mathcal{F}^\ell(s) = t$.*

Lemma 7.16 (Representation of the Product) *Let $T := T_{\mathcal{I}\times\mathcal{I}}$ be a block cluster tree based on the cluster tree $T_{\mathcal{I}}$. For each leaf $r \times t \in T$ on level ℓ and all $j = 0, \ldots, \ell$ we define*

$$\mathcal{U}(r \times t, j) := \left\{ s \in T_{\mathcal{I}} \mid \mathcal{F}^j(r) \times s \in T, s \times \mathcal{F}^j(t) \in T \text{ and at least one of the two is a leaf} \right\}.$$

Then for two matrices $A, B \in \mathcal{H}(T,k)$ and each $r \times t \in \mathcal{L}(T \cdot T)$ there holds

$$\mathcal{I} = \dot{\bigcup_{j=0,\ldots,\ell}} \dot{\bigcup_{s\in\mathcal{U}(r\times t,j)}} \hat{s} \tag{7.5}$$

$$(AB)|_{\hat{r}\times\hat{t}} = \sum_{j=0}^{\ell} \sum_{s\in\mathcal{U}(r\times t,j)} A|_{\hat{r}\times\hat{s}} B|_{\hat{s}\times\hat{t}}. \tag{7.6}$$

Proof: (**Disjointness of $\mathcal{U}(r \times t, j)$**) The elements of $\mathcal{U}(r \times t, j)$ are nodes of the cluster tree $T_{\mathcal{I}}$ on the same level j and Lemma 6.20 yields the disjointness.

(**Disjointness w.r.t.** j) Let $s_1 \in \mathcal{U}(r \times t, j_1)$ and $s_2 \in \mathcal{U}(r \times t, j_2)$, $j_2 \leq j_1$ and $\hat{s}_1 \cap \hat{s}_2 \neq \emptyset$. We want to prove $\hat{s}_1 = \hat{s}_2$. Since $s_1, s_2 \in T_{\mathcal{I}}$ we get $\hat{s}_1 \subset \hat{s}_2$ and $\mathcal{F}^{j_2}(s_1) = s_2$. It follows

$$\widehat{\mathcal{F}^{j_1}(r)} \times \hat{s}_1 \subset \widehat{\mathcal{F}^{j_2}(r)} \times \hat{s}_2, \qquad \hat{s}_1 \times \widehat{\mathcal{F}^{j_1}(t)} \subset \hat{s}_2 \times \widehat{\mathcal{F}^{j_2}(t)}. \tag{7.7}$$

Due to the definition of $\mathcal{U}(r \times t, j_2)$ at least one of $\mathcal{F}^{j_2}(r) \times s_2$ or $s_2 \times \mathcal{F}^{j_2}(t)$ is a leaf. Hence, one inclusion in (7.7) becomes an equality and $\hat{s}_1 = \hat{s}_2$.

(**Covering**) Let $j \in \mathcal{I}$. If we define $t_0 := \mathcal{F}^0(r) \times \mathcal{F}^0(r)$ and $t_0' := \mathcal{F}^0(r) \times \mathcal{F}^0(t)$, then it holds

$$t_0 \in T, \quad t_0' \in T \quad \text{and} \quad j \in \mathcal{F}^0(r).$$

If neither t_0 nor t_0' is a leaf, then there exists $s \in \operatorname{sons}(\mathcal{F}^0(r))$ such that $j \in \hat{s}$ and $t_1 := \mathcal{F}^1(r) \times s \in T$, $t_1' := s \times \mathcal{F}^1(t) \in T$. By induction we define $t_i := \mathcal{F}^i(r) \times s$, $t_i' := s \times \mathcal{F}^i(t)$ with $j \in \hat{s}$. Let i be the first index for which either $t_i = \mathcal{F}^i(r) \times s$ or $t_i' = s \times \mathcal{F}^i(t)$ is a leaf. Then $j \in \hat{s}, s \in \mathcal{U}(r \times t, i)$. ∎

Theorem 7.17 (Structure of the Product) *Let $T := T_{\mathcal{I}\times\mathcal{I}}$ be a block cluster tree based on $T_{\mathcal{I}}$ with sparsity constant C_{sp} and depth p. The exact multiplication is a mapping $\cdot : \mathcal{H}(T,k) \times \mathcal{H}(T,k) \to \mathcal{H}(T \cdot T, \tilde{k})$ for some $\tilde{k}$ which can be bounded by*

$$\tilde{k} \leq (p+1) C_{\mathrm{sp}} \max\{k, n_{\min}\}. \tag{7.8}$$

The exact multiplication can be performed with complexity

$$N_{\mathcal{H},\cdot}(T,k) \leq 4 C_{\mathrm{sp}} \max\{k, n_{\min}\} (p+1) N_{\mathrm{St}}(T,k).$$

Proof: (Rank) Let $A, B \in \mathcal{H}(T,k)$ and $r \times t \in \mathcal{L}(T \cdot T, \ell)$. Due to (7.6), we can express the product by $(p+1)\max_{j=0}^{\ell} \#\mathcal{U}(r \times t, j)$ addends, each of which is a product of two matrices. From the definition of $\mathcal{U}(r \times t, j)$ we get that for each addend one of the factors corresponds to a leaf and so its rank is bounded by $\max\{k, n_{\min}\}$. Hence, each addend has a rank bounded by $\max\{k, n_{\min}\}$. It follows that $\tilde{k} \leq (p+1)\max_{j=0}^{\ell} \#\mathcal{U}(r \times t, j) \max\{k, n_{\min}\}$. The cardinality of $\mathcal{U}(r \times t, j)$ is bounded by

$$\#\mathcal{U}(r \times t, j) \quad \leq \quad \#\{s \in T_{\mathcal{I}} \mid \mathcal{F}^j(r) \times s \in T\} \leq C_{\mathrm{sp}}(T)$$

which yields $\#\mathcal{U}(r \times s, j) \leq C_{\mathrm{sp}}(T)$.

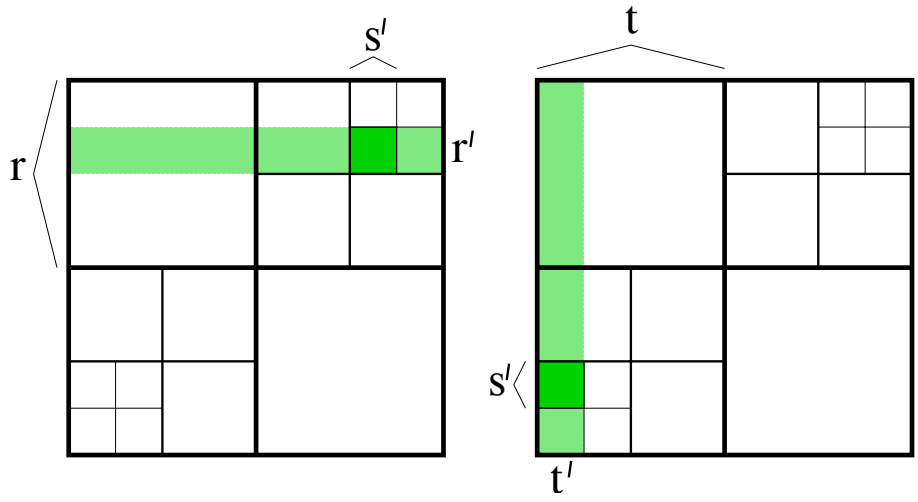


Figure 7.1: The idempotency constant $C_{\mathrm{id}}(r \times t)$ of the leaf $r \times t$ is 9.

(Complexity) Each leaf $r \times s \in \mathcal{L}(T, \ell)$ is multiplied with at most every other block $s \times t \in T^{(\ell)}$ and vice versa every leaf $s \times t \in \mathcal{L}(T, \ell)$ is multiplied with at most every other block $r \times s \in T^{(\ell)}$. The cost for the multiplication is bounded according to Lemma 7.7 by $2\kappa N_{\mathcal{H},St}(T_{s\times t}, k), \kappa := \max\{n_{\min}, k\}$. This sums up to

$$\begin{aligned} N_{\mathcal{H},\cdot}(T,k) &\leq \sum_{\ell=0}^{p} \sum_{r\times s\in\mathcal{L}(T,\ell)} \sum_{s\times t\in T^{(\ell)}} 2\kappa N_{\mathcal{H},St}(T_{s\times t},k) + \sum_{\ell=0}^{p} \sum_{s\times t\in\mathcal{L}(T,\ell)} \sum_{r\times s\in T^{(\ell)}} 2\kappa N_{\mathcal{H},St}(T_{r\times s},k) \\ &\leq 2C_{\mathrm{sp}} \sum_{\ell=0}^{p} \sum_{s\times t\in T^{(\ell)}} 2\kappa N_{\mathcal{H},St}(T_{s\times t},k) \\ &\leq 4C_{\mathrm{sp}}(p+1)\kappa N_{\mathcal{H},St}(T,k). \end{aligned}$$

■

The representation formula of the previous theorem is based on the product tree $T \cdot T$ which may differ from T. The formatted multiplication has to map into $\mathcal{H}(T, k)$ such that we have to truncate a matrix from $\mathcal{H}(T \cdot T, \tilde{k})$ to $\mathcal{H}(T, k)$. The complexity will depend upon the discrepancy between $T \cdot T$ and T. The trivial case would be an idempotent tree T in the sense $T \cdot T = T$. Sadly, block cluster trees will in general not be idempotent. The distance of $T \cdot T$ from T is measured by the idempotency constant defined next.

Definition 7.18 (Idempotency) *Let $T := T_{\mathcal{I}\times\mathcal{I}}$ be a block cluster tree based on $T_{\mathcal{I}}$. We define the elementwise idempotency (cf. Figure 7.1) $C_{\mathrm{id}}(r \times t)$ and* idempotency constant *$C_{\mathrm{id}}(T)$ by*

$$\begin{aligned} C_{\mathrm{id}}(r\times t) &:= \#\{r'\times t' \mid r'\in S^*(r), t'\in S^*(t) \text{ and } \exists s'\in T_{\mathcal{I}} : r'\times s'\in T, s'\times t'\in T\}, \\ C_{\mathrm{id}}(T) &:= \max_{r\times t\in\mathcal{L}(T)} C_{\mathrm{id}}(r\times t). \end{aligned}$$

If the tree T is fixed, the short notation C_{id} is used instead of $C_{\mathrm{id}}(T)$.

Theorem 7.19 (Formatted Multiplication) *Let $T := T_{\mathcal{I}\times\mathcal{I}}$ be a block cluster tree with idempotency constant C_{id}, sparsity constant C_{sp} and depth p. We assume (for simplicity) $n_{\min} \leq k$. The exact multiplication is a mapping $\cdot : \mathcal{H}(T,k) \times \mathcal{H}(T,k) \to \mathcal{H}(T,\tilde{k})$ with some $\tilde{k}$ bounded by*

$$\tilde{k} \leq C_{\mathrm{id}} C_{\mathrm{sp}}(p+1)k.$$

The formatted multiplication $\odot^{\mathrm{best}} : \mathcal{H}(T,k) \times \mathcal{H}(T,k) \to \mathcal{H}(T,k')$ *for any* $k' \in \mathbb{N}_0$ *is defined as the exact multiplication followed by the truncation* $\mathcal{T}_{k'}$ *of Lemma 6.12 and can be computed with complexity*

$$N_{\odot,\mathrm{best}}(T,k) \le 47 C_{\mathrm{id}}^3 C_{\mathrm{sp}}^3 k^3 (p+1)^3 \max\{\#\mathcal{I}, \#\mathcal{L}(T)\}$$

by truncating the exact product. Using the fast truncation $\mathcal{T}'_{k'}$ *of Defintion 6.17, the complexity can be reduced to*

$$N_{\odot}(T,k) \le 60 C_{\mathrm{sp}}^2 C_{\mathrm{id}} k^2 (p+1)^2 \#\mathcal{I} + 184 C_{\mathrm{sp}} C_{\mathrm{id}} k^3 (p+1) \#\mathcal{L}(T).$$

We call this mapping $\odot$ *or* $\odot^{\mathrm{fast}}$ *in contrast to* $\odot^{\mathrm{best}}$ *from above.*

Proof: (a. Rank) Due to (7.8), in each leaf of $T \cdot T$ the rank is bounded by $(p+1)C_{\mathrm{sp}}k$. If a leaf from T is contained in a leaf from $T \cdot T$, then the restriction to the leaf from T does not increase the rank. If a leaf from T contains leaves from $T \cdot T$ then their number is bounded by C_{id} and therefore the rank at most $\tilde{k}$.

(b. Complexity) We split the cost estimate into three parts: N_{mul} for calculating the exact product in $T \cdot T$, N_- for converting the `rkmatrix` blocks corresponding to leaves $\mathcal{L}^-(T)$ to `fullmatrix` format and N_+, N_+^{fast} for the (fast) truncation of the `rkmatrix` blocks to leaves $\mathcal{L}^+(T)$ with rank k'.

(b1. N_{mul}**)** According to Theorem 7.17 and Lemma 7.6, the exact product using the `rkmatrix` representation with rank $\tilde{k}$ in each leaf can be computed with complexity $8C_{\mathrm{sp}}^2(p+1)^2k^2\#\mathcal{I}$.

(b2. N_-**)** In the leaves $r \times t \in \mathcal{L}^-(T)$ we have to change the representation to `fullmatrix` format which has a cost of $2\tilde{k}\#\hat{r}\#\hat{t}$:

$$\begin{aligned} N_- &\le \sum_{r\times t\in\mathcal{L}^-(T)} 2\tilde{k}\#\hat{t}\#\hat{s} \\ &\le \sum_{r\times t\in\mathcal{L}(T)} 2\tilde{k} n_{\min}(\#\hat{r} + \#\hat{t}) \\ &\overset{L.7.4}{\le} 4C_{\mathrm{sp}}\tilde{k} n_{\min}(p+1)\#\mathcal{I} \\ &\le 4C_{\mathrm{sp}}^2 C_{\mathrm{id}}(p+1)^2 k^2 \#\mathcal{I}. \end{aligned}$$

(b3. N_+**)** For each leaf in $\mathcal{L}^+(T)$ we truncate the `rkmatrix` block of rank $\tilde{k}$ to rank k using Lemma 7.9 for the truncation or Lemma 7.10 for the fast truncation:

$$\begin{aligned} N_+ &\overset{\text{Lem.7.9}}{\le} 6\tilde{k} N_{\mathrm{St}}(T,\tilde{k}) + 23(\tilde{k})^3 \#\mathcal{L}(T) \\ &\overset{\text{Lem.7.6}}{\le} 12 C_{\mathrm{sp}}^3 C_{\mathrm{id}}^2 k^2 (p+1)^3 \#\mathcal{I} + 23 C_{\mathrm{sp}}^3 C_{\mathrm{id}}^3 k^3 (p+1)^3 \#\mathcal{L}(T) \\ &\le 35 C_{\mathrm{sp}}^3 C_{\mathrm{id}}^3 k^3 (p+1)^3 \max\{\#\mathcal{I}, \#\mathcal{L}(T)\}, \end{aligned}$$

$$\begin{aligned} N_+^{\mathrm{fast}} &\overset{\text{Lem.7.10}}{\le} C_{\mathrm{sp}} C_{\mathrm{id}} (p+1) \left(24 k N_{\mathrm{St}}(T,k) + 184 k^3 \#\mathcal{L}(T)\right) \\ &\overset{\text{Lem.7.6}}{\le} 48 C_{\mathrm{sp}}^2 C_{\mathrm{id}} k^2 (p+1)^2 \#\mathcal{I} + 184 C_{\mathrm{sp}} C_{\mathrm{id}} k^3 (p+1) \#\mathcal{L}(T). \end{aligned}$$

■

7.2.4 Inversion

The formatted inversion for matrices in `supermatrix` format was defined by use of the formatted multiplication and addition. The complexity analysis is **not** done in this way. Instead, we observe that the multiplication and inversion procedure perform the same kind of operations - only in different order.

Theorem 7.20 (Formatted Inversion) *Let* $T := T_{\mathcal{I}\times\mathcal{I}}$ *be a block cluster tree. We assume that for the* `fullmatrix` *blocks* $r \times t \in \mathcal{L}^-(T)$ *the complexity of the inversion is bounded by the complexity of the multiplication (in the case* $n_{\min} = 1$ *both are one elementary operation). Then the complexity* $N_{\mathcal{H},\mathrm{Inv}}(T,k)$ *of the formatted inversion (Algorithm 5) in the set* $\mathcal{H}(T,k)$ *is bounded by* $N_{\odot}(T,k)$.

Proof: We prove the statement by induction over the depth p of the tree T. For $p = 0$, we have assumed that the inversion is of the same complexity as the multiplication. Now let $p > 0$ and let the matrix $M \in \mathcal{H}(T, k)$ be of the block structure

$$M = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix}, \qquad \operatorname{sons}(\mathcal{I}) = \{\mathcal{I}_1, \mathcal{I}_2\}.$$

The multiplication $X := M \odot M$ involves the operations

$$X = \begin{bmatrix} M_{11} \odot M_{11} \oplus M_{12} \odot M_{21} & M_{21} \odot M_{11} \oplus M_{22} \odot M_{21} \\ M_{11} \odot M_{12} \oplus M_{12} \odot M_{22} & M_{22} \odot M_{22} \oplus M_{21} \odot M_{12} \end{bmatrix}. \tag{7.9}$$

By induction the complexity $N_{\mathcal{H},\mathrm{Inv}}(T_{\mathcal{I}_1\times\mathcal{I}_1}, k)$ and $N_{\mathcal{H},\mathrm{Inv}}(T_{\mathcal{I}_2\times\mathcal{I}_2}, k)$ for the formatted inversions of $\mathtt{m}_{11} = M_{11}$ and $\mathtt{m}_{22} = M_{22} \ominus M_{21} \odot \mathrm{Inv}_{\mathcal{H}}(M_{11}) \odot M_{12}$ (line 3 and 7 in Algorithm 5) is bounded by $N_{\mathcal{H},\odot}(T_{\mathcal{I}_1\times\mathcal{I}_1}, k)$ and $N_{\mathcal{H},\odot}(T_{\mathcal{I}_2\times\mathcal{I}_2}, k)$. The other six combinations of formatted multiplications (line 4–6 and 8–10) are of the form $M_{ij} \odot M_{j\ell}$ and appear also in (7.9). The two formatted additions (line 6 and 9) for the matrices M_{11} and M_{22} are also present in (7.9). ∎

7.3 Sparsity and Idempotency of the Block Cluster Tree $T_{\mathcal{I}\times\mathcal{I}}$

7.3.1 Construction of the Cluster Tree $T_{\mathcal{I}}$

Before we estimate the idempotency and sparsity of the tree $T_{\mathcal{I}\times\mathcal{I}}$, we will first recapitulate the construction of the cluster tree $T_{\mathcal{I}}$ by geometrically balanced splitting of the space.

Let $\mathcal{I}$ be any fixed (finite) index set and $d \in \mathbb{N}$. The basis functions associated to the indices $i \in \mathcal{I}$ map from $\mathbb{R}^d$ to $\mathbb{R}$ and have a small support Ω_i. Let m_i be any point from the support Ω_i (for nodal based basis functions one can take the nodal point).

In order to simplify the notation and visualisation we will only present the case $d = 2$. The generalisation to the case $d > 2$ is straightforward.

Construction 7.21 (Geometrically Balanced Clustering) *Without loss of generality we assume that the domain Ω is contained in the square $[0, H) \times [0, H)$. The root $\operatorname{root}(T_{\mathcal{I}})$ of the cluster tree has the whole index set $\mathcal{I}$ as a label.*

The splitting of a cluster t with corresponding box $[a, c) \times [d, f)$ is done geometrically balanced, i.e., for the midpoints $b := (a + c)/2$ and $e := (d + f)/2$ the four sons $\{s_1, s_2, s_3, s_4\}$ of t are

$$\begin{aligned} \hat{s}_1 &:= \{i \in \hat{t} \mid m_i \in [a, b) \times [d, e), \\ \hat{s}_2 &:= \{i \in \hat{t} \mid m_i \in [b, c) \times [d, e), \\ \hat{s}_3 &:= \{i \in \hat{t} \mid m_i \in [a, b) \times [e, f), \\ \hat{s}_4 &:= \{i \in \hat{t} \mid m_i \in [b, c) \times [e, f). \end{aligned}$$

If $\hat{t}$ contains less or equal to $n_{\min}$ indices then we do not split t any further. Note that the boxes $[a, b) \times [d, e)$ are passed on to the sons such that **they may differ from the bounding box**. *The box corresponding to a node $t \in T_{\mathcal{I}}$ is denoted by C_t.*

The structure of the cluster tree $T_{\mathcal{I}}$ corresponds to the regular structure of the geometrically splitting of $[0, H] \times [0, H]$. This regular structure is essential in order to bound the sparsity and idempotency of the block cluster tree to be constructed. Trees that have a large idempotency or sparsity may still be useful and allow for fast formatted arithmetics, but the bounds for the complexity are not easy to establish.

Since only the vertex m_i from Ω_i is contained in the respective box, a box B_t containing Ω_t has to be $\max_{i\in\hat{t}} \operatorname{diam}(\Omega_i)$ larger than C_t. We define the local meshwidth h_t and box B_t by

$$h_t := \max_{i\in\hat{t}} \operatorname{diam}(\Omega_i), \qquad B_t := C_t + [-h_t, h_t]^2. \tag{7.10}$$

Lemma 7.22 *For any two nodes $t, s \in T_{\mathcal{I}}^{(\ell)}$ there holds*

$$\begin{aligned} \Omega_t &\subset B_t \\ \operatorname{diam}(B_t) &= \sqrt{2}(2^{-\ell} + 2h_t)H \qquad (7.11) \\ \operatorname{dist}(B_t, B_s) &\geq \operatorname{dist}(C_t, C_s) - \sqrt{2}(h_t + h_s). \qquad (7.12) \end{aligned}$$

Proof: Let $i \in \hat{t}$. By Construction 7.21 we get $m_i \in C_t$. The elements in the support Ω_i have a distance of at most h_t from m_i and thus $\Omega_i \subset B_t$. The second and third part follow from the definition of B_t. ∎

7.3.2 Construction of the Block Cluster Tree $T_{\mathcal{I}\times\mathcal{I}}$

Based on the cluster tree $T_{\mathcal{I}}$ from Construction 7.21 and the admissibility condition (2.4) we define the block cluster tree $T := T_{\mathcal{I}\times\mathcal{I}}$ as follows.

A product index set $\hat{r} \times \hat{s}$ with corresponding boxes C_r and C_s is called admissible, if

$$\min\{\widetilde{\operatorname{diam}}(r), \widetilde{\operatorname{diam}}(s)\} \leq \eta \widetilde{\operatorname{dist}}(r, s), \qquad (7.13)$$

where the modified distance and diameter are

$$\begin{aligned} \widetilde{\operatorname{diam}}(t) &:= \operatorname{diam}(B_t), \\ \widetilde{\operatorname{dist}}(r, s) &:= \operatorname{dist}(B_r, B_s). \end{aligned}$$

If a product $\hat{r} \times \hat{s}$ is admissible with respect to (7.13) then the corresponding domain $\Omega_r \times \Omega_s$ is admissible with respect to the standard admissibility condition (2.4).

This modified admissibility condition is used to construct the block cluster tree $T_{\mathcal{I}\times\mathcal{I}}$ as usual.

Construction 7.23 (Block Cluster Tree $T_{\mathcal{I}\times\mathcal{I}}$) *Let the cluster tree $T_{\mathcal{I}}$ be given. We define the block cluster tree $T := T_{\mathcal{I}\times\mathcal{I}}$ by $\widetilde{\operatorname{root}(T)} := \mathcal{I} \times \mathcal{I}$ and for each vertex $r \times s \in T$ the set of successors*

$$\operatorname{sons}(r \times s) := \begin{cases} \{r' \times s' \mid r' \in \operatorname{sons}(r), s' \in \operatorname{sons}(s)\} & \text{if } \#\hat{r} > n_{\min} \text{ and } \#\hat{s} > n_{\min} \\ & \text{and } r \times s \text{ is inadmissible,} \\ \emptyset & \text{otherwise.} \end{cases}$$

The block cluster tree does not bear a regular structure and we have to do some work to gain the bounds for the sparsity and idempotency.

Lemma 7.24 *Let $T := T_{\mathcal{I}\times\mathcal{I}}$ be the block cluster tree of depth $p \geq 1$ built from the cluster tree $T_{\mathcal{I}}$ by Construction 7.23. Then the sparsity constant (cf. Definition 7.3) C_{sp} of T is bounded by*

$$C_{\mathrm{sp}} \leq 4 \max_{r \in T_{\mathcal{I}}} \#\{s \in T_{\mathcal{I}} \mid r \times s \in T \setminus \mathcal{L}(T)\}.$$

Proof: Let $r \in T_{\mathcal{I}}^{(\ell)}$, $\ell > 0$. Then

$$\begin{aligned} \#\{s \in T_{\mathcal{I}}^{(\ell)} \mid r \times s \in T\} &= \#\{s \in T_{\mathcal{I}}^{(\ell)} \mid \mathcal{F}(r) \times \mathcal{F}(s) \in T \setminus \mathcal{L}(T)\} \\ &\leq 4\#\{\mathcal{F}(s) \in T_{\mathcal{I}}^{(\ell-1)} \mid \mathcal{F}(r) \times \mathcal{F}(s) \in T \setminus \mathcal{L}(T)\} \\ &\leq 4\#\{s \in T_{\mathcal{I}}^{(\ell-1)} \mid \mathcal{F}(r) \times s \in T \setminus \mathcal{L}(T)\} \\ &\leq 4\#\{s \in T_{\mathcal{I}} \mid \mathcal{F}(r) \times s \in T \setminus \mathcal{L}(T)\}. \end{aligned}$$

Here we exploit that a cluster has at most 4 sons. ∎

So far, we have not posed any condition on the locality of the supports of the basis functions φ_i. If all the supports cover the whole domain Ω, then the only admissible block $\hat{t} \times \hat{s} \subset \mathcal{I} \times \mathcal{I}$ is $\hat{t} \times \hat{s} = \emptyset$. Therefore, we have to demand the locality of the supports.

Assumption 7.25 (Locality) *We assume that the supports are locally separated in the sense that there exist two constants C_{sep} and $n_{\min}$ such that*

$$\max_{i\in\mathcal{I}} \#\{j \in \mathcal{I} \mid \operatorname{dist}(\Omega_i,\Omega_j) \le C_{\mathrm{sep}}^{-1}\operatorname{diam}(\Omega_i)\} \le n_{\min}. \tag{7.14}$$

The left-hand side is the maximal number of 'rather close' supports. Note that the bound $n_{\min}$ is the same that we use for the construction of $T_{\mathcal{I}}$.

Lemma 7.26 (Sparsity, Idempotency and Depth of $T_{\mathcal{I}\times\mathcal{I}}$) *Let $h := \min_{i\in\mathcal{I}} \operatorname{diam}(\Omega_i)$. We use the same notation as in Construction 7.21 and assume that (7.14) holds for some constants $C_{\mathrm{sep}}, n_{\min}$. Let $T := T_{\mathcal{I}\times\mathcal{I}}$ be the block cluster tree from Construction 7.21 and 7.23. Then the following statements hold:*
(a) All leaves $t \times s \in \mathcal{L}(T)$ are either admissible with respect to (7.13) or $\min\{\#\hat{t}, \#\hat{s}\} \le n_{\min}$.
(b) The depth of the tree is bounded by

$$\operatorname{depth}(T) \le 1 + \log_2\left((1+2C_{\mathrm{sep}})\sqrt{2}H/h_{\min}\right).$$

(c) The sparsity constant is bounded by

$$C_{\mathrm{sp}} \le \left(2 + 8\sqrt{2}C_{\mathrm{sep}} + 4\sqrt{2}\eta^{-1}(1+2C_{\mathrm{sep}})\right)^2.$$

(d) The idempotency constant is bounded by

$$C_{\mathrm{id}} \le (2 + 4C_{\mathrm{sep}} + 2\eta(1+2\sqrt{2}C_{\mathrm{sep}}))^4$$

Proof: **(a)** Holds by Construction 7.23.

(b) Let $t \in T_{\mathcal{I}}^{(\ell)}$ be a non-leaf node, in particular $\#\hat{t} > n_{\min}$. We prove

$$h_t \le C_{\mathrm{sep}} \operatorname{diam}(C_t)$$

by contradiction: assume that there exists $i \in \hat{t}$ such that $\operatorname{diam}(\Omega_i) > C_{\mathrm{sep}} \operatorname{diam}(C_t)$. Then for all $j \in \hat{t}$

$$\operatorname{dist}(\Omega_i,\Omega_j) \le \operatorname{diam}(C_t) < C_{\mathrm{sep}}^{-1} \operatorname{diam}(\Omega_i)$$

which is a contradiction to (7.14). Using the bound on h_t we conclude

$$\operatorname{diam}(B_t) \overset{(7.11)}{=} \sqrt{2}(2^{-\ell} + 2h_t)H \overset{(7.14)}{\le} \sqrt{2}(2^{-\ell} + 2C_{\mathrm{sep}}\operatorname{diam}(C_t))H = \sqrt{2}(1+2C_{\mathrm{sep}})2^{-\ell}H \tag{7.15}$$

while $\operatorname{diam}(B_t) \ge \operatorname{diam}(\Omega_i) \ge h_{\min}$. This yields $2^{\ell} \le \sqrt{2}(1+2C_{\mathrm{sep}})H/h_{\min}$.

(c) We exploit the structure of the regular subdivision of $[0,H)\times[0,H)$ into the squares C_t. Let $t \in T_{\mathcal{I}}^{(\ell)}$ be a node with $\#\hat{t} > n_{\min}$. The number of squares C_s on level ℓ that touch C_t is at most 3^2. By induction it follows that the number of cubes on level ℓ with a distance less than $j2^{-\ell}H$ to C_t is bounded by $(1+2j)^2$. Let $s \in T_{\mathcal{I}}^{(\ell)}$ with $\#\hat{s} > n_{\min}$ and $\operatorname{dist}(C_t,C_s) > j2^{-\ell}H$. The diameter and distance of the respective bounding boxes can be estimated by

$$\begin{aligned}
\operatorname{diam}(B_t) &\overset{(7.15)}{\le} \sqrt{2}(1+2C_{\mathrm{sep}})2^{-\ell}H,\\
\operatorname{dist}(B_t,B_s) &\overset{(7.12)}{\ge} \operatorname{dist}(C_t,C_s) - \sqrt{2}(h_t+h_s)\\
&> j2^{-\ell}H - \sqrt{2}C_{\mathrm{sep}}(\operatorname{diam}(C_t)+\operatorname{diam}(C_s))\\
&= j2^{-\ell}H - \sqrt{2}C_{\mathrm{sep}}2^{1-\ell}H.
\end{aligned}$$

If $t\times s$ is not admissible with respect to (7.13) then $\min\{\operatorname{diam}(B_t),\operatorname{diam}(B_s)\} > \eta \operatorname{dist}(B_t,B_s)$. Insertion of the above estimates gives

$$\sqrt{2}(1+2C_{\mathrm{sep}})2^{-\ell}H > \eta(j2^{-\ell}H - 2\sqrt{2}C_{\mathrm{sep}}2^{-\ell}H)$$

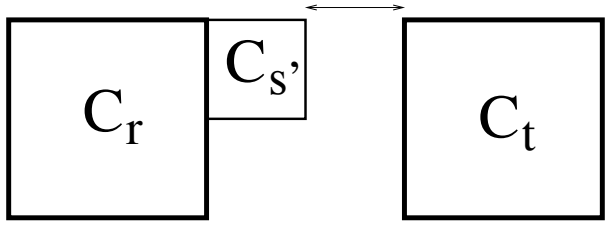


Figure 7.2: The small box $C_{s'}$ is well separated from either C_r or C_s if C_r and C_t are well separated.

which yields

$$j < \eta^{-1}\left(\sqrt{2}(1+2C_{\mathrm{sep}}) + 2\eta C_{\mathrm{sep}}\right) =: j_{\max}.$$

As a consequence the number of nodes $s \in T_{\mathcal{I}}^{(\ell)}$ not admissible to t is bounded by $(1+2j_{\max})^2$. Lemma 7.24 yields (c).

(d) Let $r \times t \in \mathcal{L}(T,\ell)$. If $\#\hat{r} \le n_{\min}$ or $\#\hat{t} \le n_{\min}$, then the elementwise idempotency is $C_{\mathrm{id}}(r\times t) = 1$. Now let $r \times t$ be admissible. Define $q := \lceil \log_2(1 + 2C_{\mathrm{sep}} + \eta(1+2\sqrt{2}C_{\mathrm{sep}}))\rceil$. We want to prove that for all vertices $r', s', t' \in T^{(\ell+q)}$, $r'\times s' \in S^*(r\times s)$ and $s'\times t' \in S^*(s\times t)$ one of the vertices $r'\times s'$ and $s'\times t'$ is a leaf. Let r', s', t' be given as above and $\min\{\#\hat{r}', \#\hat{s}', \#\hat{t}'\} > n_{\min}$.

For $u \in \{r', s', t'\}$ it holds

$$\widetilde{\mathrm{diam}}(u) \overset{(7.15)}{\le} \sqrt{2}(1+2C_{\mathrm{sep}})2^{-q-\ell}H. \tag{7.16}$$

The distance between $C_{s'}$ and $C_{r'}$ or $C_{t'}$ can be estimated (cf. Figure 7.2) by

$$\begin{aligned}\max\{\mathrm{dist}(C_{s'},C_{r'}), \mathrm{dist}(C_{s'},C_{t'})\} &\ge \mathrm{dist}(C_{r'},C_{t'}) - \mathrm{diam}(C_{s'})\\ &\ge \mathrm{dist}(B_r,B_t) - \mathrm{diam}(C_{s'}).\end{aligned}$$

Using this estimate we get

$$\begin{aligned}\eta\max\{\mathrm{dist}(B_{r'},B_{s'}), \mathrm{dist}(B_{s'},B_{t'})\} &\ge \eta\max\{\mathrm{dist}(C_{r'},C_{s'}), \mathrm{dist}(C_{s'},C_{t'})\} - \eta\max_{u\in\{r',t'\}}\sqrt{2}(h_u+h_{s'})\\ &\ge \eta\,\mathrm{dist}(B_r,B_t) - \eta\,\mathrm{diam}(C_{s'}) - \eta 2\sqrt{2}C_{\mathrm{sep}}\,\mathrm{diam}(C_{s'})\\ &\ge \min\{\mathrm{diam}(B_r),\mathrm{diam}(B_t)\} - \eta(1+2\sqrt{2}C_{\mathrm{sep}})\,\mathrm{diam}(C_{s'})\\ &\ge \mathrm{diam}(C_r) - \eta(1+2\sqrt{2}C_{\mathrm{sep}})\,\mathrm{diam}(C_{s'})\\ &= \sqrt{2}2^{-\ell}H - \eta(1+2\sqrt{2}C_{\mathrm{sep}})\sqrt{2}2^{-\ell-q}H.\end{aligned} \tag{7.17}$$

The admissiblity condition for either $r'\times s'$ or $s'\times t'$ is according to (7.16) and (7.17) fulfilled if

$$\sqrt{2}(1+2C_{\mathrm{sep}})2^{-q-\ell}H \le \sqrt{2}2^{-\ell}H - \eta(1+2\sqrt{2}C_{\mathrm{sep}})\sqrt{2}2^{-\ell-q}H,$$

which is equivalent to $(1+2C_{\mathrm{sep}})2^{-q} \le 1 - \eta(1+2\sqrt{2}C_{\mathrm{sep}})2^{-q}$, and this equation is true for the choice of $q = \lceil \log_2(1+2C_{\mathrm{sep}} + \eta(1+2\sqrt{2}C_{\mathrm{sep}}))\rceil$.

We conclude that either $r'\times s'$ or $s'\times t'$ is admissible (and has no sons). It follows that there are no vertices $r''\times s'' \in T^{(\ell+q+1)}$ and $s''\times t'' \in T^{(\ell+q+1)}$ with $r'' \in S^*(r), t'' \in S^*(t)$. Since the number of sons of a vertex is limited by 2^4, there are at most 2^{4q} vertices in $T\cdot T$ that are contained in $r\times t$. ∎

Remark 7.27 *Lemma 7.26 proves that Construction 7.21 (→ cluster tree) combined with Construction 7.23 (→ block cluster tree) yields a tree T that is sparse and idempotent with C_{sp} and C_{id} independent of the cardinality of the index set $\mathcal{I}$. The depth of the tree is estimated by the logarithm of the ratio of the smallest element to the diameter of the whole domain (which can be large). For most triangulations this ratio depends polynomially on $\#\mathcal{I}$ such that the logarithm of the ratio is proportional to* $\log(\#\mathcal{I})$.

Remark 7.28 (Admissibility for $\mathcal{H}^2$-matrices) *The results of Lemma 7.26 depend on the admissibility condition (2.4). In the context of $\mathcal{H}^2$-matrices the stronger admissibility condition*

$$\max\{\operatorname{diam}(\tau), \operatorname{diam}(\sigma)\} \leq \eta \operatorname{dist}(\tau, \sigma) \tag{7.18}$$

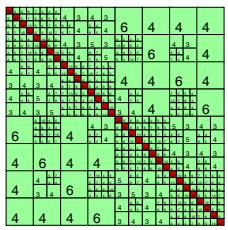
is required. The bounds for the sparsity constant C_{sp}, the idempotency constant C_{id} and the depth p of the tree also hold for this admissibility condition, because the reference cubes C_r,C_s on the same level are of equal size.

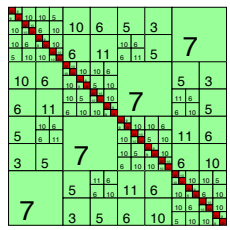
Chapter 8

Alternative Clustering Strategies

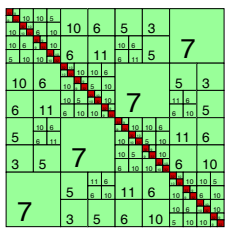
In finite element methods, the stiffness matrix is sparse but its inverse is fully populated and can be approximated by an $\mathcal{H}$-matrix. Such an approximate inverse may then be used as a preconditioner in iterative methods, for the setup and inversion of Schur complements, for the representation of matrix valued functions and so forth. Even though the complexity $\mathcal{O}(n \log^2 nk^2)$ of the $\mathcal{H}$-matrix inversion is almost linear, there are relatively large constants hidden. The following developments address this drawback successfully and allow $\mathcal{H}$-matrix based preconditioners to be competitive in the FEM context:

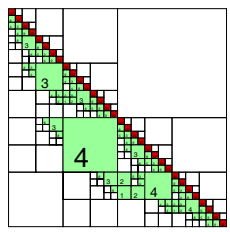
1. a weak admissibility condition yielding coarser block structures and hence yielding smaller constants in the complexity [43];





2. the introduction of an $\mathcal{H}$-LU decomposition which is computed significantly faster than an approximate inverse and provides an (in general more accurate) preconditioner [48, 33];





3. the parallelization of the $\mathcal{H}$-matrix arithmetic [46].

In this chapter, we will add a further improvement to these three components which reduces the constants in the complexity significantly: we will introduce (recursive) domain decompositions with an interior boundary, also known as *nested dissection*, into the construction of the cluster tree. This clustering algorithm, first presented in [32], will yield a block structure in which large subblocks are zero and remain zero in a subsequent LU factorization. Furthermore, the subsequent $\mathcal{H}$-LU factorization is parallelizable.

In the last part of this chapter we introduce an algebraic clustering algorithm: In previous chapters, the construction of the matrix partition of $\mathcal{H}$-matrices is based on the underlying geometry of the degrees of

freedom, i.e., information on the grid is needed in addition to the stiffness matrix itself. Here, we will develop a black box clustering algorithm that is applicable to *sparse* matrices and only needs the matrix itself as input. A matrix graph is constructed based on the sparsity structure of the matrix which is required for the subsequent algebraic clustering algorithm. We therefore obtain an algorithm for an algebraic $\mathcal{H}$-matrix construction that can be compared to algebraic multigrid (AMG) techniques [17, 54, 18, 37].

8.1 Nested Dissection

Most direct methods for sparse linear systems perform an LU factorization of the original matrix after some reordering of the indices in order to reduce fill-ins. One such popular reordering method is the so-called *nested dissection* method which exploits the concept of separation. The main idea is to separate the vertices in a (matrix) graph into three parts, two of which have no coupling between each other. The third one, referred to as an interior boundary or separator, contains couplings with (possibly both of) the other two parts. The nodes of the separated parts are numbered first and the nodes of the separator are numbered last. This process is then repeated recursively in each subgraph. An illustration of the resulting sparsity pattern is shown in Figure 8.1 for the first two decomposition steps. In domain decomposition terminology,

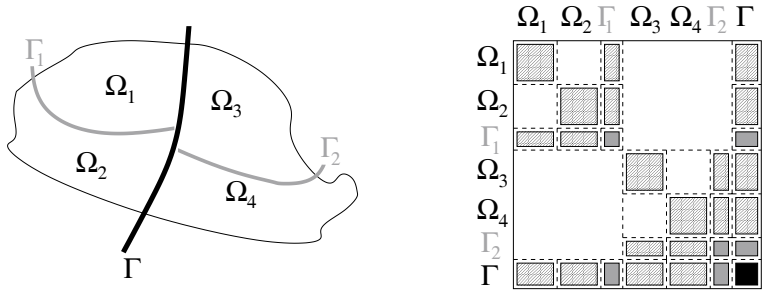


Figure 8.1: Nested dissection and resulting matrix sparsity structure.

we recursively subdivide the domain into an interior boundary and the resulting two disjoint subdomains.

A favorable property of such an ordering is that a subsequent LU factorization maintains a major part of this sparsity structure, i.e., there occurs no fill-in in the large, off-diagonal zero matrix blocks. In order to obtain a (nearly) optimal complexity, we approximate the nonzero, off-diagonal blocks in the $\mathcal{H}$-matrix representation and compute them using $\mathcal{H}$-matrix arithmetic. The blocks on the diagonal and their LU factorizations will be stored as full matrices.

8.2 Clustering based on Nested Dissection

In [41], a direct domain decomposition method is combined with the hierarchical matrix technique. In particular, a domain Ω is subdivided into p subdomains and an interior boundary Γ which separates the subdomains as shown in Figure 8.2. Within each subdomain, standard $\mathcal{H}$-matrix techniques can be used, i.e., $\mathcal{H}$-matrices constructed by the standard bisection index clustering with zero or two successors. Thus, the *first* step is the decomposition of the domain into a fixed number of subdomains, and in a *second* step, $\mathcal{H}$-matrix techniques are applied within each subdomain.

8.2.1 Clustering Based on Bisection

The standard construction of the cluster tree $T_{\mathcal{I}}$ is based on variants of binary space partitioning. The basic idea to construct the clusters is to subdivide a cluster v with support Ω_v into smaller clusters v_1, v_2 as follows:

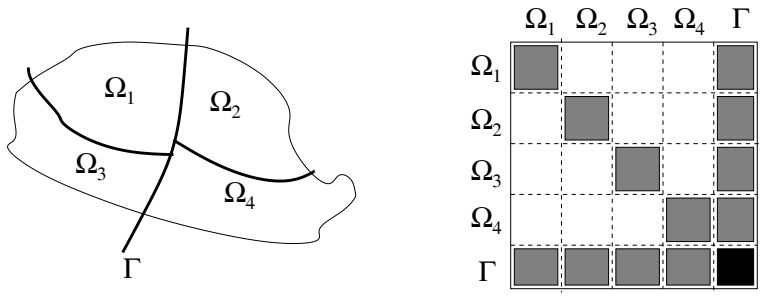


Figure 8.2: Direct domain decomposition and the resulting matrix sparsity structure

1. Let Q_v denote a box that contains all nodal points $(x_i)_{i\in v}$. For the root cluster this could be the bounding box $Q_{\mathcal{I}} := B_{\mathcal{I}}$.
2. Subdivide the box Q_v into two parts $Q_v = Q_1 \dot\cup Q_2$ of equal size.
3. Define the set $\mathrm{sons}(v) = \{v_1, v_2\}$ of v by

$$v_1 := \{i \in v \mid x_i \in Q_1\}, \quad v_2 := \{i \in v \mid x_i \in Q_2\}$$

and use the boxes $Q_{v_1} := Q_1$, $Q_{v_2} := Q_2$ for the further splitting of the sons.

The subdivision is typically performed such that the resulting diameters of the boxes associated with successor clusters become as small as possible so that clusters eventually become separated and fulfill the standard admissibility condition. A visualization of this process, the geometric regular bisection, is given in Figure 8.3.

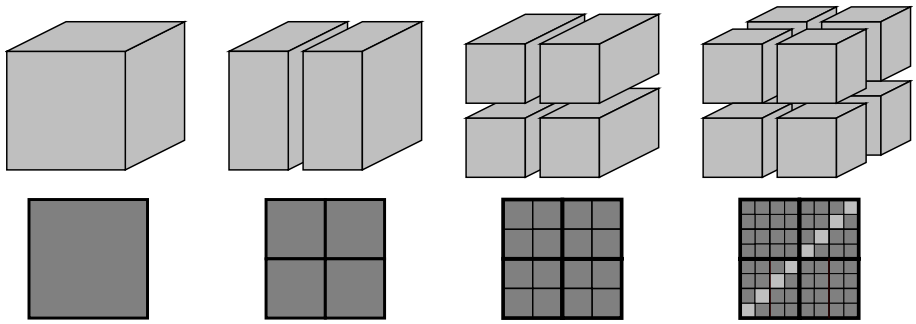


Figure 8.3: The box $Q_{\mathcal{I}}$ that contains the whole domain $\Omega = [0,1]^3$ is successively subdivided into two subboxes. The corresponding matrix A displays a hierarchical 2×2 block-structure. The eight light grey blocks are not standard admissible but "weakly" admissible.

8.2.2 Clustering Based on Domain Decomposition

A new construction of the cluster tree is based on the decomposition of a cluster v with corresponding domain Ω_v into three sons, i.e., $\mathrm{sons}(v) = \{v_1, v_2, v_3\}$, where v_1 and v_2 contain the indices of the two disconnected subdomains Ω_{v_1} and Ω_{v_2} while $v_3 = v \setminus (v_1 \cup v_2)$ contains the indices corresponding to the separator $\Gamma_v = \Omega_{v_3}$, cf. Figure 8.1.

Due to the fact that the distance between Ω_{v_1} and Ω_{v_2}, given by the width of the separator Γ_v, is typically very small compared to the diameters of Ω_{v_1}, Ω_{v_2}, the block cluster $v_1 \times v_2$ is neither standard admissible nor weakly admissible. However, since the corresponding matrix block is zero and remains zero during an

LU factorization, we want to regard it as admissible; in fact, we can assign a fixed rank of zero to this block. This means that an admissibility condition suitable for domain decomposition based block clusters has to distinguish between the sets of domain-clusters $\mathcal{C}_{\mathrm{dom}}$ and interface-clusters $T_{\mathcal{I}} \setminus \mathcal{C}_{\mathrm{dom}}$ which will be specified in Construction 8.2.

Definition 8.1 (DD-admissibility) *Let $T_{\mathcal{I}}$ be a cluster tree for the index set $\mathcal{I}$ and let $\mathcal{C}_{\mathrm{dom}} \subset T_{\mathcal{I}}$ be the subset of domain-clusters which will be defined in Construction 8.2. We define the DD-admissibility condition by*

$$\mathrm{Adm}_{\mathrm{DD}}(s \times t) = \texttt{true} \quad :\Leftrightarrow \quad (s \neq t,\ s,t \in \mathcal{C}_{\mathrm{dom}}) \quad \textit{or} \quad s \times t \textit{ standard admissible.} \tag{8.1}$$

The formal DD-clustering process is presented in the following Construction 8.2. A visualization for the clustering of a three-dimensional domain and the resulting block structure is presented in Figure 8.4.

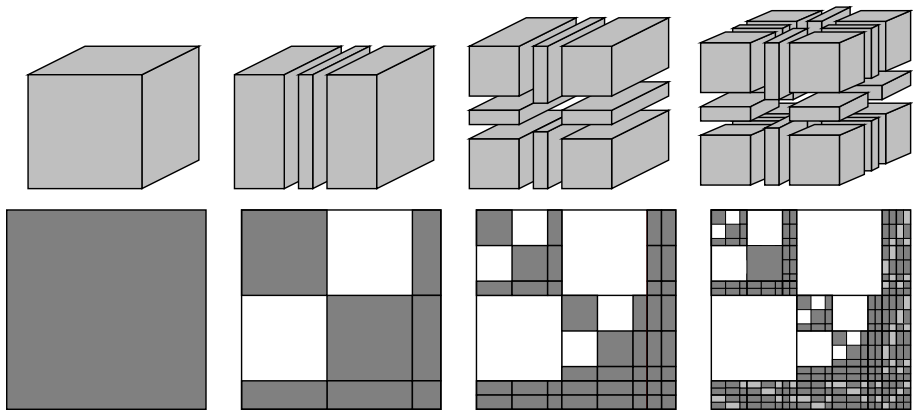


Figure 8.4: The domain $\Omega = [0,1]^3$ is successively subdivided using DD-clustering. The top row displays the respective bounding boxes used for the subdivision. The bottom row shows the resulting hierarchical block structure of the matrix A. The light grey blocks are still weakly admissible (boxes intersect in at most one corner) whereas the white blocks satisfy $\mathrm{Adm}_{\mathrm{DD}}$.

Construction 8.2 (DD-clustering, domain-cluster, interface-cluster) *The cluster tree $T_{\mathcal{I}}$ as well as the set of domain-clusters $\mathcal{C}_{\mathrm{dom}}$ and interface-clusters $T_{\mathcal{I}} \setminus \mathcal{C}_{\mathrm{dom}}$ is constructed recursively, starting with*

- *the root $\mathcal{I}$, $\mathcal{C}_{\mathrm{dom}} := \{\mathcal{I}\}$ and*
- *the bounding box $Q_I := B_{\mathcal{I}}$ that contains the domain Ω.*

Clusters that satisfy $\#v \leq n_{min}$ will be no further refined. For all other clusters we distinguish between domain-clusters and interface-clusters. If a cluster v and a box $Q_v = \bigotimes_{i=1}^{d} [\alpha_i, \beta_i]$ satisfying $x_j \in Q_v$ for all $j \in v$ are given, we introduce new boxes Q_1 and Q_2 by splitting Q_v in half in the coordinate direction i_{split} of maximal extent.

Domain-clusters*: For $v \in \mathcal{C}_{\mathrm{dom}}$, we define the three successors*

$$v_1 := \{i \in v \mid x_i \in Q_1\}, \quad v_2 := \{i \in v \mid \mathrm{supp}\varphi_i \cap \Omega_{v_1} = \emptyset\}, \quad v_3 := v \setminus (v_1 \cup v_2) \tag{8.2}$$

and correspondingly $S(v) := \{v_1, v_2, v_3\}$, $\mathcal{C}_{\mathrm{dom}} := \mathcal{C}_{\mathrm{dom}} \cup \{v_1, v_2\}$. To the domain-clusters v_1 and v_2 we associate the boxes $Q_{v_1} := Q_1$ and $Q_{v_2} := Q_2$. The interface-cluster is equipped with the "flat" box $Q_{v_3} := \bigotimes_{i=1}^{d} [\tilde{\alpha}_i, \tilde{\beta}_i]$ where $\tilde{\alpha}_i := \alpha_i$ and $\tilde{\beta}_i := \beta_i$ except for the splitting coordinate $i = i_{\mathrm{split}}$ where we set

$$\tilde{\alpha}_i := \frac{1}{2}(\alpha_i + \beta_i) - h_{v_3}, \quad \tilde{\beta}_i := \frac{1}{2}(\alpha_i + \beta_i) + h_{v_3}, \quad h_{v_3} := \max_{j \in v_3} \mathrm{diam}(\mathrm{supp}\,\varphi_j).$$

Interface-clusters: *We define the interface-level of a cluster,* $\mathrm{level}_{\mathrm{int}}(v)$*, as the distance of* v *to the nearest domain-cluster in the cluster tree. For* $v \in T_{\mathcal{I}} \setminus \mathcal{C}_{\mathrm{dom}}$ *and associated flat box* Q_v*, we (possibly) split the box* Q_v *into two boxes* $Q_v = Q_1 \dot\cup Q_2$ *in a direction* j *different from the flat direction* i_{split}*. More precisely, the set of sons is defined by*

$$S(v) := \begin{cases} \{v\} & \mathrm{level}_{\mathrm{int}}(v) \equiv 0 (\mathrm{mod}\ d), \\ \{\{i \in v \mid x_i \in Q_1\},\ \{i \in v \mid x_i \in Q_2\}\} & \textit{otherwise.} \end{cases} \tag{8.3}$$

The associated boxes of the sons are $Q_{v_i} := Q_i$*.*

A visualization of the DD-clustering is given in Figure 8.4 for the first three subdivision steps (top). The corresponding block cluster tree and partition of the matrix (bottom) is constructed in the canonical way. The characteristic sparsity pattern can be noticed already after the first subdivision step. Since the dimension is $d = 3$ in this example, the interface-clusters that occur after the first and second subdivision have interface-levels $\mathrm{level}_{\mathrm{int}}(v) \in \{1, 2\}$. Only the four "vertical" interface-clusters that are present after the third subdivision satisfy $\mathrm{level}_{\mathrm{int}}(v) = 3$ and will not be subdivided in the next subdivision step according to (8.3).

Remark 8.3 *1. The construction of the cluster tree is guided by the fact that matrix entries* A_{ij} *equal zero if the corresponding supports of basis functions are disjoint. One can therefore replace the condition "*$\mathrm{supp}\varphi_i \cap \Omega_{v_1} = \emptyset$*" in (8.2) by "*$A_{ij} = 0$ *for all* $j \in v_1$*".*

2. The subdivision of interface-clusters $v \in T_{\mathcal{I}} \setminus \mathcal{C}_{\mathrm{dom}}$ *is delayed every d-th step in order to calibrate the diameters of interface-clusters with those of domain-clusters. Without this calibration, the subdivision of the largest interface cluster will reach the leaves after* $\approx \log_2(N^{1-1/d}) = (1 - 1/d)\log_2(N)$ *steps while the depth of the cluster tree is approximately* $\log_2(N)$*. This would lead to undesirable fill-in in the* $\mathcal{H}$*-matrix so that we would lose the almost linear complexity and instead get the same complexity as the classical nested dissection.*

8.3 Black Box Clustering for Sparse Matrices

Hierarchical matrices are based on a block cluster tree $T_{\mathcal{I}\times\mathcal{I}}$ that describes the (hierarchical) partition of a matrix into admissible and inadmissible blocks. The (formatted) arithmetic in the $\mathcal{H}$-matrix format is defined in an abstract setting where only this partition is needed but not the geometric information by which the cluster tree $T_{\mathcal{I}}$ was built. However, for the construction of the bock cluster tree $T_{\mathcal{I}\times\mathcal{I}}$ we use the cluster tree $T_{\mathcal{I}}$ and the admissibility condition which both require information about the distance between (geometric) entities as well as the diameters of (geometric) entities.

In some applications, the geometric information describing the support of the basis functions that are used for the discretization of the partial differential operator might not be available. Instead, only the already assembled sparse stiffness matrix A is given.

In this case, our goal is to derive a black-box clustering algorithm and an admissibility condition that use only the sparse stiffness matrix A as input. In the following, both the cluster tree $T_{\mathcal{I}}$ as well as the block cluster tree $T_{\mathcal{I}\times\mathcal{I}}$ will be derived from A so that we can solve the linear system $Ax = b$ either directly (by an almost exact $\mathcal{H}$-LU decomposition) or by use of an $\mathcal{H}$-LU preconditioner in some iterative method. Since the only information available is the sparse matrix A itself, we use the matrix graph $\mathcal{G}(A)$ in order to redefine the notions of distances and diameters.

The symmetrised graph $\mathcal{G}(A) = (V, E)$ of a matrix $A \in \mathbb{R}^{\mathcal{I}\times\mathcal{I}}$ is defined by the vertex and edge sets

$$V := \mathcal{I}, \qquad E := \{(i, j) \in \mathcal{I} \times \mathcal{I} \mid (A_{ij} \neq 0 \vee A_{ji} \neq 0)\}. \tag{8.4}$$

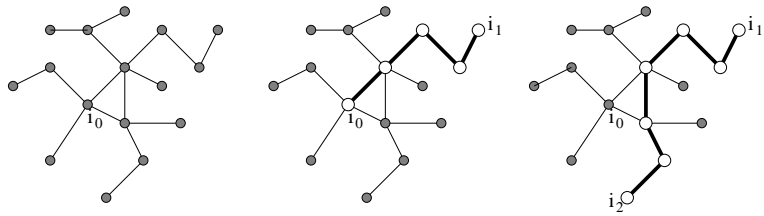
The distance between two nodes $i, j \in \mathcal{I}$ in a connected matrix graph, defined as the length of the shortest path between these two nodes, can be computed in $\mathcal{O}(N)$, $N := \#\mathcal{I}$, by Dijkstra's algorithm. However, computing all pairwise distances would require quadratic complexity $\mathcal{O}(N^2)$ which is prohibitively expensive. For the construction of the cluster tree $T_{\mathcal{I}}$ it is sufficient to split a set $v \subset \mathcal{I}$ into two subsets v_1, v_2 (and

possibly the interface v_3). The complexity of the splitting should not exceed $\mathcal{O}(\#v)$, so that the complexity of the setup of the cluster tree will be $\mathcal{O}(N \log N)$.

Construction 8.4 (Black Box DD-Clustering) *Let $\mathcal{G}_{\mathrm{sym}}(A)$ denote the symmetrised matrix graph of a sparse matrix $A \in \mathbb{R}^{\mathcal{I}\times\mathcal{I}}$. We assume $\mathcal{G}_{\mathrm{sym}}(A)$ to be connected. The root of the cluster tree $T_{\mathcal{I}}$ is* $\mathrm{root}(T_{\mathcal{I}}) := \mathcal{I}$. *Initially, we set $\mathcal{C}_{\mathrm{dom}} := \{\mathcal{I}\}$.*

We define the cluster tree $T_{\mathcal{I}}$ recursively by the son relation of an arbitrary cluster $v \subset \mathcal{I}$. Let $\bar{v} \supset v$ denote a connected superset of v; for all domain-clusters (e.g. the root $\mathcal{I}$), this is $\bar{v} := v$.

1. *We determine two nodes $i_1, i_2 \in v$ with almost maximal distance in $\bar{v}$ as follows: We choose an arbitrary vertex $i_0 \in v$, determine a node $i_1 \in v$ with the largest distance to i_0 in $\bar{v}$ and then determine a node $i_2 \in v$ with the largest distance to i_1 in $\bar{v}$:*



2. *We determine the distance of all $i \in \bar{v}$ to both i_1 and i_2 in $\bar{v}$. Subsequently, we partition $\bar{v}$ into two parts, each of which is again connected:*

$$\widehat{v}_1 := \{i \in \bar{v} \mid \mathrm{dist}_{\bar{v}}(i, i_1) \leq \mathrm{dist}_{\bar{v}}(i, i_2)\}, \quad \widehat{v}_2 := \bar{v} \setminus \widehat{v}_1.$$

3. *For a domain-cluster $v \in \mathcal{C}_{\mathrm{dom}}$, we partition v into the three parts*

$$v_1 := \widehat{v}_1, \quad v_2 := \{i \in v \mid \mathrm{dist}_{\bar{v}}(i, i_2) < \mathrm{dist}_{\bar{v}}(i, i_1) - 1\}, \quad v_3 := v \setminus (v_1 \cup v_2).$$

   The corresponding supersets are $\bar{v}_1 := v_1$, $\bar{v}_2 := v_2$, $\bar{v}_3 := \bar{v}$, the sons are $S(v) := \{v_1, v_2, v_3\}$ with two domain-clusters $\mathcal{C}_{\mathrm{dom}} := \mathcal{C}_{\mathrm{dom}} \cup \{v_1, v_2\}$ and one interface-cluster v_3. For the recursion, we first subdivide the two domain-clusters v_1, v_2 and afterwards the interface-cluster v_3. To calibrate the sizes of interface-clusters with those of domain-clusters (see Remark 8.3), we define the target depth $d(v_3) := \max\{\mathrm{depth}(T_{v_1}),\ \mathrm{depth}(T_{v_2})\}$, the average reduction factor $\rho(v_3) := (n_{\min}/\#v_3)^{1/p}$ and the target size $\mathrm{s}(v_3) := \#v_3\rho(v_3)$ for the next subdivision step.

4. *For an interface-cluster $v \notin \mathcal{C}_{\mathrm{dom}}$, we distinguish between two cases:*

   - *If the target depth is $d(v) = 0$, then the recursion stops, i.e., $S(v) := \emptyset$.*
   - *If the target depth is $d(v) > 0$ and the size of v is less than the target size, $\#v < \mathrm{s}(v)$, then $S(v) := \{v\}$, $\mathrm{s}(v) := \mathrm{s}(v)\rho(v)$ and $d(v) := d(v) - 1$. Proceed with the son of v.*
   - *If the target depth is $d(v) > 0$ and the size of v is larger than the target size, $\#v \geq \mathrm{s}(v)$, then we define*

$$v_1 := v \cap \widehat{v}_1, \quad v_2 := v \cap \widehat{v}_2.$$

     The corresponding supersets are $\bar{v}_1 := \widehat{v}_1$, $\bar{v}_2 := \widehat{v}_2$, the sons are $S(v) := \{v_1, v_2\}$ with two interface-clusters v_1, v_2. The target sizes of both v_1 and v_2 are $\mathrm{s}(v_1) := \mathrm{s}(v_2) := \mathrm{s}(v)\rho(v)$ and $\rho(v_1) := \rho(v_2) := \rho(v)$, and the target depths are $d(v_1) := d(v_2) := d(v)$.

Remark 8.5 (Black Box DD-Admissibility) *The black box DD-admissibility is defined by estimating the diameter and distance in the matrix graph. The diameter of a cluster v we estimate by $\widetilde{\mathrm{diam}}(v) := \mathrm{dist}(i_0, i_1) + \mathrm{dist}(i_1, i_2)$, where i_0, i_1, i_2 are defined as above. The admissibility $\mathrm{diam}(v) \leq \eta\widetilde{\mathrm{dist}}(v, w)$ can be tested checking all nodes with a distance of at most $\eta^{-1}\,\mathrm{diam}(v)$ from v. If one of these nodes belongs to w, then the block is inadmissible, otherwise it is admissible.*

Chapter 9

$\mathcal{H}^2$-matrices

In order to find a hierarchical matrix approximation of an integral operator, we have used a degenerate expansion of the kernel function. We have constructed this expansion by applying interpolation to either the first or the second argument of the kernel function.

In this chapter, we will take a closer look at the properties of this expansion and derive a specialized variant of hierarchical matrices that can be treated by more efficient algorithms.

$\mathcal{H}^2$-matrices were introduced in [45], where the approximation was based on Taylor expansions. We will use the construction described in [25, 10], which is based on interpolation.

9.1 Motivation

We consider the bilinear form

$$a(u,v) = \int_\Omega v(x) \int_\Omega g(x,y)u(y)\,\mathrm{d}y\,\mathrm{d}x$$

corresponding to an integral operator. In Chapter 3, we have replaced the original kernel function $g(\cdot,\cdot)$ by a degenerate approximation

$$\tilde{g}^{t,s}(x,y) := \sum_{\nu\in K} g(x_\nu^t, y)\mathcal{L}_\nu^t(x)$$

for admissible cluster pairs (t,s) (we assume that $\operatorname{diam}(Q^t) \leq \operatorname{diam}(Q^s)$ holds for all cluster pairs in the block cluster tree, so that we can always interpolate in the x-variable without losing the approximation property).

Discretizing this approximated kernel function leads to the representation

$$\begin{aligned}
\tilde{G}_{ij}^{t,s} &= \int_\Omega \varphi_i(x) \int_\Omega \tilde{g}^{t,s}(x,y)\varphi_j(y)\,\mathrm{d}y\,\mathrm{d}x \\
&= \sum_{\nu\in K} \int_\Omega \varphi_i(x) \int_\Omega g(x_\nu^t, y)\mathcal{L}_\nu^t(x)\varphi_j(y)\,\mathrm{d}y\,\mathrm{d}x \\
&= \sum_{\nu\in K} \left(\int_\Omega \varphi_i(x)\mathcal{L}_\nu^t(x)\,\mathrm{d}x\right)\left(\int_\Omega \varphi_j(y) g(x_\nu^t, y)\,\mathrm{d}y\right) \\
&= \left(A^{t,s}B^{t,s^\top}\right)_{ij}
\end{aligned}$$

for all $i \in \hat{t}$ and $j \in \hat{s}$, where

$$A_{i\nu}^{t,s} = \int_\Omega \varphi_i(x)\mathcal{L}_\nu^t(x)\,\mathrm{d}x \quad \text{and} \quad B_{j\nu}^{t,s} = \int_\Omega \varphi_j(y) g(x_\nu^t, y)\,\mathrm{d}y.$$

Let us take a closer look at the matrix $A^{t,s}$: since we integrate a Lagrange polynomial corresponding to a cluster t multiplied with basis functions corresponding to the same cluster, this matrix depends only on the cluster t, but not on the cluster s. This means that for each cluster $t \in T_{\mathcal{I}}$, we have a matrix $V^t \in \mathbb{R}^{\hat{t}\times K}$ defined by

$$V^t_{i\nu} := \int_\Omega \varphi_i(x)\mathcal{L}^t_\nu(x)\,\mathrm{d}x \tag{9.1}$$

for $i \in \hat{t}$ and $\nu \in K$ satisfying $V^t = A^{t,s}$ for *all* admissible leaves (t,s) of the block cluster tree $T_{\mathcal{I}\times\mathcal{I}}$. We define

$$R^t := \{s \in T_{\mathcal{I}} \ : \ (t,s) \in \mathcal{L}^+(T_{\mathcal{I}\times\mathcal{I}})\},$$

i.e., R^t contains all clusters s such that (t,s) is an admissible leaf of the block cluster tree $T_{\mathcal{I}\times\mathcal{I}}$, and rewrite our observation in the form

$$A^{t,s} = V^t \qquad \text{for all } s \in R^t.$$

This equation has two major implications: we can save memory, since we have to store V^t only once, and we can save time in the matrix-vector multiplication, since due to

$$\sum_{s\in R^t} A^{t,s}{B^{t,s}}^\top x|_{\hat{s}} = \sum_{s\in R^t} V^t {B^{t,s}}^\top x|_{\hat{s}} = V^t \left(\sum_{s\in R^t} {B^{t,s}}^\top x|_{\hat{s}}\right),$$

we can sum over vectors of length $k = \#K$ instead of over vectors of length $\#\hat{t}$.

But there is more: consider a second cluster $t' \in T_{\mathcal{I}}$. Since we use the same space of polynomials for *all* clusters, we have

$$\operatorname{span}\{\mathcal{L}^t_\nu \ : \ \nu \in K\} = \operatorname{span}\{\mathcal{L}^{t'}_{\nu'} \ : \ \nu' \in K\},$$

so there must be coefficients $T^{t'}_{\nu'\nu} \in \mathbb{R}$ such that

$$\mathcal{L}^t_\nu = \sum_{\nu'\in K} T^{t'}_{\nu'\nu}\mathcal{L}^{t'}_{\nu'} \tag{9.2}$$

holds, i.e., we can represent the Lagrange polynomials corresponding to the cluster t by the Lagrange polynomials corresponding to the cluster t'. Since we are dealing with Lagrange polynomials, the computation of the coefficients $T^{t'}_{\nu'\nu}$ is especially simple: they are given by

$$T^{t'}_{\nu'\nu} = \mathcal{L}^t_\nu(x^{t'}_{\nu'}). \tag{9.3}$$

For each index $i \in \hat{t}$, we can find a $t' \in \operatorname{sons}(t)$ with $i \in \hat{t}'$, and equation (9.2) implies

$$V^t_{i\nu} = \int_\Omega \varphi_i(x)\mathcal{L}^t_\nu(x)\,\mathrm{d}x = \sum_{\nu'\in K} T^{t'}_{\nu'\nu}\int_\Omega \varphi_i(x)\mathcal{L}^{t'}_{\nu'}(x)\,\mathrm{d}x = \sum_{\nu'\in K} T^{t'}_{\nu'\nu}V^{t'}_{i\nu'} = (V^{t'}T^{t'})_{i\nu}. \tag{9.4}$$

This equation allows us to speed up the matrix-vector multiplication even more: computing $V^t y^t$ directly for a vector $y^t \in \mathbb{R}^K$ requires $\mathcal{O}(k\#\hat{t})$ operations. If t is not a leaf, i.e., if $\operatorname{sons}(t) \neq \emptyset$, there is a $t' \in \operatorname{sons}(t)$ for each $i \in \hat{t}$ such that $i \in \hat{t}'$, and this implies $(V^t y^t)_i = (V^{t'}T^{t'}y^t)_i$. So instead of computing $V^t y^t$ directly, we can compute $T^{t'}y^t$ for all sons $t' \in \operatorname{sons}(t)$, and this will only require $\mathcal{O}(k^2)$ operations.

9.2 $\mathcal{H}^2$-matrices

9.2.1 Uniform $\mathcal{H}$-matrices

We have seen that the matrices $A^{t,s}$ which appear in our degenerate approximation of the admissible matrix blocks have many desirable properties due to the fact that they are discretizations of Lagrange polynomials. Unfortunately, the matrices $B^{t,s}$ are not of this kind.

In order to fix this, we change our approximation scheme: instead of applying interpolation only to the x coordinate, we apply it to both coordinates:

$$\tilde{g}^{t,s}(x,y) := (\mathfrak{I}_m^t \otimes \mathfrak{I}_m^s)[g](x,y) = \sum_{\nu\in K}\sum_{\mu\in K} g(x_\nu^t, x_\mu^s)\mathcal{L}_\nu^t(x)\mathcal{L}_\mu^s(y). \tag{9.5}$$

Discretizing this approximation of the kernel, we find

$$\begin{aligned}\tilde{G}_{ij} &:= \int_\Omega \varphi_i(x)\int_\Omega \tilde{g}^{t,s}(x,y)\varphi_j(y)\,\mathrm{d}y\,\mathrm{d}x\\ &= \sum_{\nu\in K}\sum_{\mu\in K} g(x_\nu^t,x_\mu^s)\underbrace{\left(\int_\Omega \varphi_i(x)\mathcal{L}_\nu^t(x)\,\mathrm{d}x\right)}_{=V_{i\nu}^t}\underbrace{\left(\int_\Omega \varphi_j(y)\mathcal{L}_\mu^s(y)\,\mathrm{d}y\right)}_{=V_{j\mu}^s} = V^t S^{t,s} V^{s\top}\end{aligned} \tag{9.6}$$

for $i\in\hat{t}$ and $j\in\hat{s}$, where $S^{t,s}\in\mathbb{R}^{K\times K}$ is defined by

$$S_{\nu\mu}^{t,s} := g(x_\nu^t, x_\mu^s). \tag{9.7}$$

Now, we have what we need: the matrices V^t and V^s have the desired properties, and the matrix $S^{t,s}$ is only of dimension $k\times k$ (for $k := \#K$).

Exercise 6 (Error bound for the symmetric kernel approximation) *Assume that*

$$\max\{\operatorname{diam}(Q_t),\operatorname{diam}(Q_s)\} \le \eta\operatorname{dist}(Q_t,Q_s) \tag{9.8}$$

holds. Prove that

$$|g(x,y)-\tilde{g}^{t,s}(x,y)| \le \frac{Cd(m+1)^{2d-1}}{2^{2m}\operatorname{dist}(Q_t,Q_s)^\sigma}(c_0\eta)^{m+1}$$

holds for all $x\in Q_t$ and $y\in Q_s$ under the assumptions from Subsection 3.3.3.

Definition 9.1 (Cluster basis) *Let $T_\mathcal{I}$ be a cluster tree for the index set $\mathcal{I}$. A family $(V^t)_{t\in T_\mathcal{I}}$ of matrices is a* cluster basis*, if for each $t\in T_\mathcal{I}$ there is a finite index set K^t such that $V^t\in\mathbb{R}^{\hat{t}\times K^t}$.*

A cluster basis $(V^t)_{t\in T_\mathcal{I}}$ is of constant order*, if there is a set K such that $K^t = K$ holds for each $t\in T_\mathcal{I}$.*

Obviously, the matrices V^t introduced by (9.1) form a constant-order cluster basis.

Definition 9.2 (Uniform $\mathcal{H}$-matrix) *Let $T_{\mathcal{I}\times\mathcal{J}}$ be a block cluster tree and let $V = (V^t)_{t\in T_\mathcal{I}}$ and $W = (W^s)_{s\in T_\mathcal{J}}$ be cluster bases for the index sets $\mathcal{I}$,$\mathcal{J}$. We define the set of* uniform $\mathcal{H}$-matrices *with row basis V and column basis W as*

$$\begin{aligned}\mathcal{H}(T_{\mathcal{I}\times\mathcal{J}},V,W) := \{M\in\mathbb{R}^{\mathcal{I}\times\mathcal{J}} \mid\ &\text{for all admissible } (t,s)\in\mathcal{L}(T_{\mathcal{I}\times\mathcal{J}}),\\ &\text{there is } S^{t,s}\in\mathbb{R}^{K^t\times K^s} \text{ with } M|_{\hat{t}\times\hat{s}} = V^t S^{t,s} W^{s\top}\}.\end{aligned}$$

The matrices $S^{t,s}$ are called coupling matrices*.*

A uniform $\mathcal{H}$-matrix is of constant order*, if the cluster bases $(V^t)_{t\in T_\mathcal{I}}$ and $(W^s)_{s\in T_\mathcal{J}}$ are of constant order.*

Remark 9.3 *Differently from $\mathcal{H}$-matrices, the set $\mathcal{H}(T_{\mathcal{I}\times\mathcal{J}},V,W)$ of uniform $\mathcal{H}$-matrices for fixed bases V and W is a* subspace *of $\mathbb{R}^{\mathcal{I}\times\mathcal{J}}$.*

Exercise 7 (Strongly admissible block cluster tree) *In order to get a good approximation by an $\mathcal{H}^2$-matrix, we have to replace the admissibility condition* (2.4) *by the strong admissibility condition* (9.8)*.*

Modify the function `build_supermatrix_from_cluster2` *from Exercise 2 in such a way that it uses the strong admissibility condition* (9.8) *instead of the standard admissibility.*

Exercise 8 ($\mathcal{H}^2$-matrix) *Modify* `build_supermatrix_from_cluster2` *from Exercise 7 in such a way that for admissible blocks, a* `uniformmatrix` *is created instead of an* `rkmatrix`*. Since you need a* `clusterbasis` *to create a* `uniformmatrix`*, you have to modify the parameters for your function:*

```
psupermatrix
build_supermatrix_from_cluster2(pclusterbasis row, pclusterbasis col,
                                double eta);
```

Each `supermatrix` *corresponds to a pair $t \times s$ of clusters. Make sure that the fields* `row` *and* `column` *of the* `supermatrix` *point to the cluster bases corresponding to the correct clusters.*

Due to $\operatorname{rank}(V^t S^{t,s} W^{s\top}) \leq \operatorname{rank}(S^{t,s}) \leq \min\{\#K^t, \#K^s\}$, each uniform $\mathcal{H}$-matrix is also an $\mathcal{H}$-matrix with rank $k := \max\{\#K^t, \#K^s \ : \ t \in T_{\mathcal{I}}, s \in T_{\mathcal{J}}\}$.

Using the kernel approximation (9.5) in each admissible block will lead to blocks satisfying (9.6), this matrix is a uniform $\mathcal{H}$-matrix of constant order with cluster bases defined by (9.1) and coefficient matrices defined by (9.7).

Multiplying a vector $x \in \mathbb{R}^{\mathcal{I}}$ can be organized in a procedure consisting of four steps:

1. **Forward transformation:** Compute $x^s := {W^s}^\top x|_{\hat{s}}$ for all clusters $s \in T_{\mathcal{I}}$.
2. **Multiplication:** Compute
$$y^t := \sum_{s \in R^t} S^{t,s} x^s$$
for all clusters $t \in T_{\mathcal{I}}$.
3. **Backward transformation:** Compute $y \in \mathbb{R}^{\mathcal{I}}$ defined by
$$y_i := \sum_{t, i \in \hat{t}} (V^t y^t)_i.$$
4. **Non-admissible blocks:** Treat non-admissible blocks as in the case of standard $\mathcal{H}$-matrices.

Lemma 9.4 (Complexity of the multiplication phase) *Let C_{sp} be the sparsity constant of $T_{\mathcal{I}\times\mathcal{I}}$. For a constant-order cluster basis with $k = \#K$, the multiplication phase requires $\mathcal{O}(C_{\mathrm{sp}} k^2 \#T_{\mathcal{I}})$ operations.*

Proof: By definition, we have $\#R^t \leq C_{\mathrm{sp}}$. Since the multiplication by $S^{t,s}$ requires $\mathcal{O}(k^2)$ operations and has to be performed for each $s \in R^t$, the computation of y^t for a cluster $t \in T_{\mathcal{I}}$ requires $\mathcal{O}(C_{\mathrm{sp}} k^2)$ operations.

Summing over all clusters concludes the proof. ∎

Lemma 9.5 (Complexity of the multiplication for non-admissible blocks) *Let C_{sp} be the sparsity constant of $T_{\mathcal{I}\times\mathcal{I}}$, and let $n_{\min} := \max\{\#\hat{t} \mid t \in \mathcal{L}(T_{\mathcal{I}})\}$. We assume for all inadmissible leaves $b = (t, s) \in \mathcal{L}^-(T_{\mathcal{I}\times\mathcal{I}})$ of the block cluster tree, both t and s are leaves of the cluster tree, i.e., $t, s \in \mathcal{L}(T_{\mathcal{I}})$ (cf. Exercise 2). Then the treatment of the non-admissible blocks in the matrix-vector multiplication requires $\mathcal{O}(C_{\mathrm{sp}} n_{\min}^2 \#T_{\mathcal{I}})$ operations.*

Proof: There are not more than $\#T_{\mathcal{I}}$ leaves, so there are not more than $C_{\mathrm{sp}} \#T_{\mathcal{I}}$ leaf nodes in the block cluster tree. Let $b = (t, s) \in \mathcal{L}(T_{\mathcal{I}\times\mathcal{I}})$ be one of these leaf nodes. By assumption, t and s must be leaf clusters of $T_{\mathcal{I}}$, so we have $\max\{\#\hat{t}, \#\hat{s}\} \leq n_{\min}$, and the multiplication by the corresponding full matrix can be completed in $\mathcal{O}(n_{\min}^2)$ operations. ∎

In typical applications, we have $\#T_{\mathcal{I}} \leq n$, so the second and fourth step of the matrix-vector multiplication can be accomplished in linear complexity with respect to the number of degrees of freedom n.

Unfortunately, a naive implementation of the remaining steps, namely the forward and backward transformation, requires $\mathcal{O}(n(p+1)k)$ operations, where p is the depth of $T_{\mathcal{I}}$, and due to $p \approx \log(n)$, we would only reach almost linear complexity in n. In order to be able to perform matrix-vector multiplications in *optimal* complexity with respect to n, we need to improve these transformations.

9.2.2 Nested cluster bases

Definition 9.6 (Nested cluster basis) *A cluster basis* $(V^t)_{t\in T_\mathcal{I}}$ *is* nested, *if for each non-leaf cluster* $t \in T_\mathcal{I}$ *and each son cluster* $t' \in \operatorname{sons}(t)$, *there is a* transfer matrix $T^{t'} \in \mathbb{R}^{K^{t'}\times K^t}$ *satisfying*

$$(V^t y^t)_i = (V^{t'} T^{t'} y^t)_i \tag{9.9}$$

for all vectors $y^t \in \mathbb{R}^{K^t}$ *and all indices* $i \in \hat{t}'$.

Example 9.7 (Two sons) *If we assume that* $\operatorname{sons}(t) = \{t_1, t_2\}$ *with* $t_1 \neq t_2$, *we have* $\hat{t} = \hat{t}_1 \dot\cup \hat{t}_2$, *and (9.9) takes the form*

$$V^t = \begin{pmatrix} V^{t_1} T^{t_1} \\ V^{t_2} T^{t_2} \end{pmatrix} = \begin{pmatrix} V^{t_1} & \\ & V^{t_2} \end{pmatrix} \begin{pmatrix} T^{t_1} \\ T^{t_2} \end{pmatrix}.$$

Applying this equation recursively, we find

$$\operatorname{range}(V^t|_{\hat{t}'\times K^t}) \subseteq \operatorname{range}(V^{t'})$$

for all clusters $t' \in T_\mathcal{I}$ *with* $\hat{t}' \subseteq \hat{t}$.

Definition 9.8 ($\mathcal{H}^2$-matrix) *A uniform $\mathcal{H}$-matrix whose column and row cluster basis are nested is called an* $\mathcal{H}^2$-matrix.

Due to equation (9.4), the cluster basis defined by (9.1) is nested.

Let us consider the backward transformation: for $i \in \mathcal{I}$, we have to compute

$$y_i := \sum_{s, i\in\hat{s}} (V^s y^s)_i.$$

We single out a non-leaf cluster $t \in T_\mathcal{I}$ and apply (9.9) to the cluster $t' \in \operatorname{sons}(t)$ with $i \in \hat{t}'$:

$$\begin{aligned} y_i &= \sum_{s\neq t, i\in\hat{s}} (V^s y^s)_i + (V^t y^t)_i = \sum_{s\neq t, i\in\hat{s}} (V^s y^s)_i + (V^{t'} T^{t'} y^t)_i \\ &= \sum_{s\neq t, s\neq t', i\in\hat{s}} (V^s y^s)_i + V^{t'}(y^{t'} + T^{t'} y^t)_i. \end{aligned}$$

The cluster t does no longer appear in this sum, since its contribution has been added to the vectors corresponding to its son. This means that we can define a new set $(\hat{y}^s)_{s\in T_\mathcal{I}}$ by setting

$$\hat{y}^s := \begin{cases} 0 & \text{if } s = t, \\ y^s + T^s y^t & \text{if } s \in \operatorname{sons}(t), \\ y^s & \text{otherwise,} \end{cases}$$

and find

$$y_i = \sum_{s\ni i} (V^s y^s)_i = \sum_{s\ni i} (V^s \hat{y}^s)_i = \sum_{s\ni i, s\neq t} (V^s \hat{y}^s)_i$$

for *all* $i \in \mathcal{I}$. We can apply this technique to recursively eliminate all non-leaf clusters:

```
void
backward_clusterbasis(pclusterbasis cb, double *y)
{
  /* ... some initialization ... */

  if(sons > 0) {
```

```
    yindex = 0;
    for(i=0; i<sons; i++) {
      addeval_lapack(son[i]->kt, kt, T[i], yt, son[i]->yt);
      backward_clusterbasis(son[i], y + yindex);
      yindex += son[i]->n;
    }
  }
  else
    addeval_lapack(n, kt, V, yt, y);
}
```

Lemma 9.9 (Complexity of the backward transformation) *For a constant-order cluster basis with $k := \#K$, the fast backward transformation requires $\mathcal{O}(kn + k^2 \#T_{\mathcal{I}}))$ operations.*

Proof: For a non-leaf cluster $t \in T_{\mathcal{I}}$, we multiply y^t by $T^{t'}$ for all of its sons $t' \operatorname{sons}(t)$. This requires $\mathcal{O}(k^2)$ operations. Since each cluster has not more than one father, not more than $\mathcal{O}(\#T_{\mathcal{I}})$ such multiplications are performed, so treating all non-leaf clusters is accomplished in $\mathcal{O}(k^2 \#T_{\mathcal{I}})$ operations.

For a leaf cluster $t \in T_{\mathcal{I}}$, we multiply y^t by V^t. This requires $\mathcal{O}(k\#\hat{t})$ operations. Since the index sets corresponding to the leaves of $T_{\mathcal{I}}$ form a partition of $\mathcal{I}$ (cf. Lemma 2.7), we have

$$\sum_{t \in \mathcal{L}(T_{\mathcal{I}})} k\#\hat{t} = kn,$$

so the backward transformation requires $\mathcal{O}(kn + k^2 \#T_{\mathcal{I}})$ operations. ■

Let us now turn our attention to the forward transformation, i.e., to the task of computing $x^s := {W^s}^\top x|_{\hat{s}}$ for all $s \in T_{\mathcal{I}}$. Let $s \in T_{\mathcal{I}}$. If $\operatorname{sons}(s) = \emptyset$, we compute x^s directly. Otherwise, we first compute $x^{s'}$ for all $s' \in \operatorname{sons}(s)$ and observe that

$$x^s = {W^s}^\top x|_{\hat{s}} = \sum_{s' \in \operatorname{sons}(s)} {T^{s'}}^\top {W^{s'}}^\top x|_{\hat{s}'} = \sum_{s' \in \operatorname{sons}(s)} {T^{s'}}^\top x^{s'}$$

holds, i.e., we can use the vectors $x^{s'}$ corresponding to the sons of s to compute x^s. The result is the following recursive procedure:

```
void
forward_clusterbasis(pclusterbasis cb, const double *x)
{
  /* ... some initialization ... */

  clear_vector(kt, xt);
  if(sons > 0) {
    xindex = 0;
    for(i=0; i<sons; i++) {
      forward_clusterbasis(son[i], x + xindex);
      addevaltrans_lapack(son[i]->kt, kt, T[i], son[i]->xt, xt);
      xindex += son[i]->n;
    }
  }
  else
    addevaltrans_lapack(n, kt, V, xt, x);
}
```

Remark 9.10 (Complexity of the forward transformation) *We can treat the forward transformation by an identical argument as in the proof of Lemma 9.9 and reach an identical bound for its the complexity.*

Remark 9.11 *In special situations, we can reduce the complexity of the forward and backward transformation to $\mathcal{O}(n)$ by using different ranks for different clusters [55, 56, 16].*

9.2.3 Implementation

The structure used for the representation of a cluster basis is similar to that used for clusters:

Implementation 9.12 (`clusterbasis`) *The* `clusterbasis` *structure is defined as follows:*

```
typedef struct _clusterbasis clusterbasis;
typedef clusterbasis *pclusterbasis;

struct _clusterbasis {
  pccluster t;

  double **T;
  double *V;

  double *xt;
  double *yt;

  int k;
  int kt;
  int n;

  int sons;
  pclusterbasis *son;
};
```

The fields `sons` *and* `son` *are used to form a tree of* `clusterbasis` *structures similar to the cluster tree.*

The entry `t` *points to the cluster this cluster basis is used for.*

The field `k` *gives the* maximal *possible rank for which memory has been allocated, while the field* `kt` *gives the* current *rank. The field* `n` *gives the number of indices in the corresponding cluster, it is identical to* `t->size`.

The array `T` *contains the transfer matrices $T^{t'}$. The entry* `T[i]` *corresponds to the* `i`*-th son* `son[i]` *and represents a matrix in* FORTRAN *format with* `son[i]->kt` *rows and* `kt` *columns.*

The array `V` *contains the matrix V^t corresponding to this cluster, stored in standard* FORTRAN *format with* `n` *rows and* `kt` *columns. Typically, this array will only be used if* `t` *is a leaf cluster.*

The fields `xt` *and* `yt` *are auxiliary variables of size* `kt` *that store the vectors x^t and y^t used in the matrix-vector multiplication algorithm.*

Using this representation of a cluster basis, the representation of a uniform $\mathcal{H}$-matrix is straightforward: we introduce a new matrix type containing the coefficient matrix $S^{t,s}$ and pointers to the cluster bases and add it to the `supermatrix` structure:

Implementation 9.13 *The structure* `uniformmatrix` *is similar to* `rkmatrix`*:*

```
typedef struct _uniformmatrix uniformmatrix;
typedef uniformmatrix *puniformmatrix;

struct _uniformmatrix {
  pclusterbasis row;
```

```
  pclusterbasis column;

  int rows;
  int cols;

  int kr;
  int kc;
  int ktr;
  int ktc;

  double *S;
};
```

The pointers `row` *and* `column` *give us the cluster basis corresponding to this block:* V^t *corresponds to* `row` *and* V^s *corresponds to* `column`*.*

The field `rows` *stores the number of rows of this matrix block, while* `cols` *stores the number of columns.*

The fields `kr` *and* `kc` *give the* maximal *ranks of the row and column basis, the field* `ktr` *and* `ktc` *give the* current *ranks.*

Finally, the array `S` *contains the coefficient matrix* $S^{t,s}$ *in standard* FORTRAN *format with* `ktr` *rows and* `ktc` *columns.*

Implementation 9.14 (Changes in `supermatrix`) *In order to be able to treat uniform* $\mathcal{H}$*-matrices and* $\mathcal{H}^2$*-matrices, we have to add three fields to the* `supermatrix` *structure:*

```
pclusterbasis row;
pclusterbasis col;
puniformmatrix u;
```

The fields `row` *and* `col` *give us the cluster bases corresponding to this supermatrix, if it describes a uniform* $\mathcal{H}$*-matrix. If it describes an admissible block of a uniform* $\mathcal{H}$*-matrix, the field* `u` *contains the corresponding coefficient matrix* $S^{t,s}$*.*

There are two ways of performing a matrix-vector multiplication by a uniform matrix: we can compute $y := V^t S^{t,s} {V^s}^\top x|_s$ directly, or we can compute only $y^t := S^{t,s} x^s$. Obviously, the second choice is much more efficient than the first. Its implementation is very simple if the LAPACK library is used:

```
void
fasteval_uniformmatrix(puniformmatrix um)
{
  eval_lapack(um->ktr, um->ktc, um->S, um->col->xt, um->row->yt);
}
```

Obviously, this routine requires the representation `um->col->xt` of the vector x^s to be already initialized. This can be done by calling `forward_clusterbasis` before calling the standard `eval_supermatrix` and by calling `backward_clusterbasis` afterwards. We have redefined `eval_supermatrix`, `addeval_supermatrix`, `evaltrans_supermatrix` and `addevaltrans_supermatrix` in such a way that the forward and backward transformations are performed automatically if a cluster basis is present, i.e., if the fields `row` and `col` of the `supermatrix` are not null pointers.

9.3 Orthogonal cluster bases

In order to simplify the conversion of an arbitrary matrix into an $\mathcal{H}^2$-matrix, we introduce a special class of cluster bases:

Definition 9.15 (Orthogonal cluster bases) *A cluster basis* $(V^t)_{t\in T_\mathcal{I}}$ *is* orthogonal, *if*

$$(V^t)^\top V^t = I$$

holds for all clusters $t \in T_\mathcal{I}$.

Before we can prove some of the properties of orthogonal cluster bases, we need some nomenclature:

Definition 9.16 (Matrix Hilbert space) *Let* $\mathcal{I}$, $\mathcal{J}$ *be arbitrary finite index sets. We define the* Frobenius inner product *on the space* $\mathbb{R}^{\mathcal{I}\times\mathcal{J}}$ *by*

$$\langle A, B\rangle_F := \sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}} A_{ij}B_{ij}$$

for $A, B \in \mathbb{R}^{\mathcal{I}\times\mathcal{J}}$. *Obviously, we have* $\langle A, A\rangle_F = \|A\|_F^2$, *so the Frobenius inner product turns the matrix space* $\mathbb{R}^{\mathcal{I}\times\mathcal{J}}$ *into a Hilbert space.*

Lemma 9.17 (Matrix products) *Let* $\mathcal{I}, \mathcal{J}, \mathcal{K}$ *be arbitrary finite index sets. Let* $A \in \mathbb{R}^{\mathcal{I}\times\mathcal{J}}$, $B \in \mathbb{R}^{\mathcal{J}\times\mathcal{K}}$ *and* $C \in \mathbb{R}^{\mathcal{I}\times\mathcal{K}}$. *Then we have*

$$\langle AB, C\rangle_F = \langle B, A^\top C\rangle_F = \langle A, CB^\top\rangle_F.$$

Proof: Trivial. ∎

Lemma 9.18 (Best approximation) *Let* $(V^t)_{t\in T_\mathcal{I}}, (W^s)_{s\in T_\mathcal{J}}$ *be orthogonal cluster bases. Let* $M \in \mathbb{R}^{\hat{t}\times\hat{s}}$ *be an arbitrary matrix. Then*

$$S_M := (V^t)^\top M W^s$$

satisfies

$$\|M - V^t S_M (W^s)^\top\|_F \leq \|M - V^t S (W^s)^\top\|_F$$

for all $S \in \mathbb{R}^{K^t\times K^s}$, *i.e.,* S_M *is the optimal coefficient matrix for representing* M *as a uniform matrix block.*

Proof: We introduce the function

$$f : \mathbb{R}^{K^t\times K^s} \to \mathbb{R}, \quad S \mapsto \frac{1}{2}\|V^t S (W^s)^\top - M\|_F^2.$$

The minimum S^* of f is the optimal coefficient matrix. Since f is a quadratic function, its minimum satisfies the equation

$$0 = Df(S^*)\cdot R = \langle V^t S^* (W^s)^\top, V^t R (W^s)^\top\rangle_F - \langle M, V^t R (W^s)^\top\rangle_F \tag{9.10}$$

for all $R \in \mathbb{R}^{\hat{t}\times\hat{s}}$, i.e.,

$$\begin{aligned}\langle S_M, R\rangle_F &= \langle (V^t)^\top M W^s, R\rangle_F = \langle M, V^t R (W^s)^\top\rangle_F \overset{(9.10)}{=} \langle V^t S^* (W^s)^\top, V^t R (W^s)^\top\rangle_F \\ &= \langle S^*, \underbrace{(V^t)^\top V^t}_{=I} R \underbrace{(W^s)^\top W^s}_{=I}\rangle_F = \langle S^*, R\rangle_F.\end{aligned}$$

Since the Frobenius inner product in non-degenerate, this implies $S^* = S_M$. ∎

This lemma implies that computing the optimal coefficient matrices is a simple task once the cluster bases have been constructed.

As an example, let us consider the conversion of an `rkmatrix` to a `uniformmatrix`: the `rkmatrix` is given in the form $M = AB^\top$, so the optimal coupling matrix

$$S_M := (V^t)^\top M W^s = ((V^t)^\top A)((W^s)^\top B)^\top$$

can be computed by the following simple procedure:

```
void
convertrk2_uniformmatrix(prkmatrix r, puniformmatrix u)
{
  /* ... some initialization ... */

  Ac = allocate_matrix(u->ktr,r->kt);
  Bc = allocate_matrix(u->ktc,r->kt);

  for(i=0; i<r->kt; i++) {
    forward_clusterbasis(u->row, r->a + i*r->rows);
    copy_lapack(u->ktr, u->row->xt, Ac + i*u->ktr);

    forward_clusterbasis(u->col, r->b + i*r->cols);
    copy_lapack(u->ktc, u->col->xt, Bc + i*u->ktc);
  }

  multrans2_lapack(u->ktr, u->ktc, r->kt,
                   Ac, Bc, u->S);

  freemem(Bc); freemem(Ac);
}
```

9.4 Adaptive cluster bases

We have seen that $\mathcal{H}^2$-matrices provide an efficient way of dealing with matrices that result from the discretization of integral operators. Now we want to examine the case of general matrices, i.e., we want to find an algorithm that approximates an arbitrary matrix into an $\mathcal{H}^2$-matrix (cf. [9, 15, 7]).

9.4.1 Matrix error bounds

Since we intend to compute row and column cluster bases separately, we need to separate the estimates for V^t and W^s:

Lemma 9.19 (Separation of row and column bases) *Let $(V^t)_{t\in T_\mathcal{I}}, (W^s)_{s\in T_\mathcal{J}}$ be orthogonal cluster bases. Let $M \in \mathbb{R}^{\hat{t}\times\hat{s}}$. Then*

$$\|M - V^t S_M (W^s)^\top\|_F^2 \leq \|M - V^t (V^t)^\top M\|_F^2 + \|M^\top - W^s (W^s)^\top M^\top\|_F^2.$$

Proof: Let $B, C \in \mathbb{R}^{\hat{t}\times\hat{s}}$. We start by observing

$$\langle B - V^t (V^t)^\top B, V^t (V^t)^\top C\rangle_F = \langle V^t (V^t)^\top B - V^t \underbrace{(V^t)^\top V^t}_{=I} (V^t)^\top B, C\rangle_F = 0. \tag{9.11}$$

Setting $B = M$ and $C = M - MW^s(W^s)^\top$, we find that we can apply Pythagoras' equation in order to prove

$$\begin{aligned}\|M - V^t S_M (W^s)^\top\|_F^2 &= \|M - V^t (V^t)^\top M + V^t (V^t)^\top (M - MW^s (W^s)^\top)\|_F^2 \\ &= \|M - V^t (V^t)^\top M\|_F^2 + \|V^t (V^t)^\top (M - MW^s (W^s)^\top)\|_F^2.\end{aligned}$$

Setting $B = C = M - MW^s(W^s)^\top$ in (9.11), we can again use Pythagoras' equation to get

$$\|B\|_F^2 = \|B - V^t (V^t)^\top B\|_F^2 + \|V^t (V^t)^\top B\|_F^2 \geq \|V^t (V^t)^\top B\|_F^2$$

and therefore

$$\|V^t(V^t)^\top(M - MW^s(W^s)^\top)\|_F^2 \leq \|M - MW^s(W^s)^\top\|_F^2.$$

Observing $\|M - MW^s(W^s)^\top\|_F = \|M^\top - W^s(W^s)^\top M^\top\|_F$ concludes the proof. ∎

Lemma 9.20 (Approximation error) *Let $(V^t)_{t\in T_\mathcal{I}}$ be orthogonal cluster bases. Let $M \in \mathbb{R}^{\hat{t}\times\hat{s}}$. Then we have*

$$\|M - V^t(V^t)^\top M\|_F^2 = \|M\|_F^2 - \|(V^t)^\top M\|_F^2.$$

Proof: Due to (9.11), we have

$$\begin{aligned}\|M - V^t(V^t)^\top M\|_F^2 &= \|M\|_F^2 + \|V^t(V^t)^\top M\|_F^2 - 2\langle M, V^t(V^t)^\top M\rangle_F^2 \\ &= \|M\|_F^2 + \|V^t(V^t)^\top M\|_F^2 - 2\langle V^t(V^t)^\top M, V^t(V^t)^\top M\rangle_F^2 = \|M\|_F^2 - \|V^t(V^t)^\top M\|_F^2.\end{aligned}$$

The orthogonality of V^t implies

$$\|V^t(V^t)^\top M\|_F^2 = \langle V^t(V^t)^\top M, V^t(V^t)^\top M\rangle_F = \langle (V^t)^\top M, \underbrace{(V^t)^\top V^t}_{=I}(V^t)^\top M\rangle_F = \|(V^t)^\top M\|_F^2,$$

which proves our claim. ∎

This means that minimizing the approximation error is equivalent to maximizing $\|(V^t)^\top M\|_F$.

9.4.2 Construction of a cluster basis for one cluster

We recall Lemma 6.4 to see that the singular value decomposition can be used to find the orthogonal low-rank matrix V^t that solves this maximization problem: we set $n := \#\hat{t}$ and compute the singular value decomposition $M = V\Sigma U^\top$ of M with orthogonal matrices $V \in \mathbb{R}^{\hat{t}\times n}$, $W \in \mathbb{R}^{\hat{s}\times n}$ and a diagonal matrix $\Sigma = \operatorname{diag}(\sigma_1, \ldots, \sigma_n) \in \mathbb{R}^{n\times n}$, where $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$. We choose a rank k and use $I^{n,k^t} \in \mathbb{R}^{n\times k^t}$ defined by

$$I_{ij}^{n,k} = \delta_{ij}$$

for $i \in \{1, \ldots, n\}$, $j \in \{1, \ldots, k\}$ to define $V^t := VI^{n,k}$ (in short: the matrix V^t consists of the first k columns of V). Now we see that

$$M - V^t(V^t)^\top M = V\Sigma U^\top - VI^{n,k}(I^{n,k})^\top V^\top V\Sigma U = V(I - I^{n,k}(I^{n,k})^\top)\Sigma U$$

holds, and since

$$I^{n,k}(I^{n,k})_{ij}^\top = \begin{cases}\delta_{ij} & \text{if } i \leq k_t \\ 0 & \text{otherwise,}\end{cases}$$

we get

$$\|M - V^t(V^t)^\top M\|_F^2 = \sum_{i=k+1}^{n} \sigma_i^2.$$

We note that

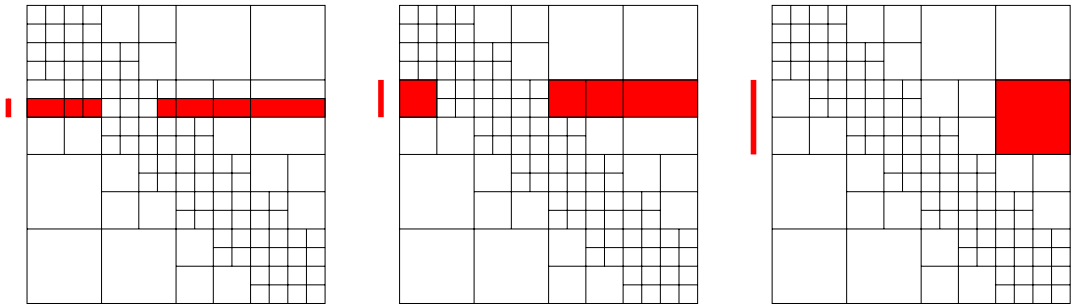
$$MM^\top = V\Sigma U^\top U\Sigma^\top V^\top = V\Sigma^2 V^\top$$

implies that, since we are not interested in the matrix U, we do not have to compute a full singular value decomposition of M, but only a Schur decomposition of the Gram matrix $MM^\top$. Then the eigenvectors will correspond to the columns of V and the eigenvalues to the squared singular values.

By definition, the matrix V^t will not only be used for one block, but for an entire block row of the matrix. The block row is given by

$$R_+^t := \{s \in T_\mathcal{J} \ : \ \text{there is } t^+ \in T_\mathcal{I} \text{ with } \hat{t} \subseteq \hat{t}^+ \text{ such that } t^+ \times s \text{ is an admissible leaf of } T_{\mathcal{I}\times\mathcal{J}}\}.$$

Due to the nested structure of cluster bases, we have to consider not only blocks directly connected to t, but also those that are connected to ancestors of t.



Block rows influencing different clusters

Using the row blocks described by R_+^t, we can now formulate the optimization problem: let $M \in \mathbb{R}^{\mathcal{I}\times\mathcal{J}}$ be a matrix. We need to find an orthogonal rank-k-matrix V^t that maximizes

$$\sum_{s\in R_+^t} \|(V^t)^\top M|_{\hat{t}\times\hat{s}}\|_F^2. \tag{9.12}$$

As we have seen before, this problem can be solved by computing the eigenvectors and eigenvalues of the Gram matrix

$$G := \sum_{s\in R_+^t} M|_{\hat{t}\times\hat{s}} M|_{\hat{t}\times\hat{s}}^\top. \tag{9.13}$$

9.4.3 Construction of a nested basis

We can apply the above construction to all clusters in $T_\mathcal{I}$ and will get a cluster basis, but this basis will in general not be nested. We have to modify the algorithm in order to ensure that it results in a nested cluster basis: For all leaf clusters, we can apply the direct construction. For non-leaf clusters $t \in T_\mathcal{I}$, we assume that we already have computed suitable orthogonal matrices $V^{t'}$ for all sons $t' \in \operatorname{sons}(t)$ and have to ensure that there are matrices $T^{t'} \in \mathbb{R}^{k^{t'}\times k}$ such that

$$V_i^t = (V^{t'}T^{t'})_i$$

holds for all $t' \in \operatorname{sons}(t)$ and $i \in \hat{t}$. In order to simplify the presentation, we will only consider the case that t has two sons, i.e., $\operatorname{sons}(t) = \{t_1, t_2\}$. Then the condition (9.9) can be written in the form

$$V^t = \begin{pmatrix} V^{t_1}T^{t_1} \\ V^{t_2}T^{t_2} \end{pmatrix}$$

and instead of finding V^t maximizing (9.12), we have to find T^{t_1} and T^{t_2} maximizing

$$\sum_{s\in R_+^t} \left\| \begin{pmatrix} V^{t_1}T^{t_1} \\ V^{t_2}T^{t_2} \end{pmatrix}^\top \begin{pmatrix} M|_{\hat{t}_1\times\hat{s}} \\ M|_{\hat{t}_2\times\hat{s}} \end{pmatrix} \right\|_F^2 = \sum_{s\in R_+^t} \left\| \begin{pmatrix} T^{t_1} \\ T^{t_2} \end{pmatrix}^\top \begin{pmatrix} (V^{t_1})^\top M|_{\hat{t}_1\times\hat{s}} \\ (V^{t_2})^\top M|_{\hat{t}_2\times\hat{s}} \end{pmatrix} \right\|_F^2$$

If we introduce the matrices

$$\widehat{M}^{t_1,s} := (V^{t_1})^\top M|_{\hat{t}_1\times\hat{s}} \quad \text{and} \quad \widehat{M}^{t_2,s} := (V^{t_2})^\top M|_{\hat{t}_2\times\hat{s}},$$

we get

$$\sum_{s\in R_+^t} \|(V^t)^\top M|_{\hat{t}\times\hat{s}}\|_F^2 = \sum_{s\in R_+^t} \left\| \begin{pmatrix} T^{t_1} \\ T^{t_2} \end{pmatrix}^\top \begin{pmatrix} \widehat{M}^{t_1,s} \\ \widehat{M}^{t_2,s} \end{pmatrix} \right\|_F^2.$$

By construction, we know that V^{t_1} and V^{t_2} are already orthogonal, and we want the matrix V^t to be orthogonal as well, i.e., to satisfy

$$I = (V^t)^\top V^t = \begin{pmatrix} V^{t_1} T^{t_1} \\ V^{t_2} T^{t_2} \end{pmatrix}^\top \begin{pmatrix} V^{t_1} T^{t_1} \\ V^{t_2} T^{t_2} \end{pmatrix} = \begin{pmatrix} T^{t_1} \\ T^{t_2} \end{pmatrix}^\top \begin{pmatrix} (V^{t_1})^\top V^{t_1} T^{t_1} \\ (V^{t_2})^\top V^{t_2} T^{t_2} \end{pmatrix} = \begin{pmatrix} T^{t_1} \\ T^{t_2} \end{pmatrix}^\top \begin{pmatrix} T^{t_1} \\ T^{t_2} \end{pmatrix}.$$

Now we can summarize the algorithm:

1. If $t \in T_{\mathcal{I}}$ is a leaf, we compute the Gram matrix

$$G := \sum_{s \in R^t_+} M|_{\hat{t} \times \hat{s}} M|_{\hat{t} \times \hat{s}}^\top.$$

   We build V^t from the eigenvectors corresponding to the k largest eigenvalues of G and set

$$\widehat{M}^{t,s} := (V^t)^\top M|_{\hat{t} \times \hat{s}}$$

   for all $s \in R^t_+$.

2. If $t \in T_{\mathcal{I}}$ is not a leaf, we compute cluster bases for the son clusters t_1, t_2 recursively and then compute the reduced Gram matrix

$$\widehat{G} := \sum_{s \in R^t_+} \begin{pmatrix} \widehat{M}^{t_1,s} \\ \widehat{M}^{t_2,s} \end{pmatrix} \begin{pmatrix} \widehat{M}^{t_1,s} \\ \widehat{M}^{t_2,s} \end{pmatrix}^\top.$$

   We build $T^{t_1,t}$ and $T^{t_2,t}$ from the eigenvectors corresponding to the k largest eigenvalues of $\widehat{G}$ and set

$$\widehat{M}^{t,s} := \begin{pmatrix} T^{t_1} \\ T^{t_2} \end{pmatrix}^\top \begin{pmatrix} \widehat{M}^{t_1,s} \\ \widehat{M}^{t_2,s} \end{pmatrix} = (T^{t_1})^\top \widehat{M}^{t_1,s} + (T^{t_2})^\top \widehat{M}^{t_2,s}$$

   for all $s \in R^t_+$.

9.4.4 Efficient conversion of $\mathcal{H}$-matrices

We can implement the basic algorithm given above directly and will get a suitable nested row cluster bases for the matrix M. Unfortunately, the computation of the Gram matrix

$$G = \sum_{s \in R^t_+} M|_{\hat{t} \times \hat{s}} M|_{\hat{t} \times \hat{s}}^\top$$

for a general matrix will require $\mathcal{O}((\#\hat{t})^2(\#\hat{s}))$ operations, leading to a total complexity of $\mathcal{O}(n^2)$. While this is acceptable for dense matrices, it is not for matrices that are already stored in $\mathcal{H}$-matrix format.

If M is an $\mathcal{H}$-matrix, then $s \in R^t_+$ implies that there is a cluster $t^+ \in T_{\mathcal{I}}$ such that $b := t^+ \times s \in T_{\mathcal{I} \times \mathcal{J}}$ is admissible, i.e., that $M|_{\hat{t}^+ \times \hat{s}} = AB^\top$ holds for a rank parameter $k_{\mathcal{H}} \in \mathbb{N}$ and matrices $A \in \mathbb{R}^{\hat{t}^+ \times k_{\mathcal{H}}}$, $B \in \mathbb{R}^{\hat{s} \times k_{\mathcal{H}}}$. Therefore we have

$$M|_{\hat{t} \times \hat{s}} M|_{\hat{t} \times \hat{s}}^\top = A|_{\hat{t} \times k} B^\top B A|_{\hat{t} \times k}^\top = A|_{\hat{t} \times k} G_b A|_{\hat{t} \times k}^\top. \tag{9.14}$$

If we have prepared the matrix

$$G_b := B^\top B,$$

in advance, we can compute the product (9.14) in $\mathcal{O}((\hat{t})^2 k_{\mathcal{H}})$ operations, which leads to a total complexity of $\mathcal{O}(n(k^2 + k_{\mathcal{H}})p)$. The preparation of all matrices G_b can be accomplished in $\mathcal{O}(nk_{\mathcal{H}}^2 p)$ operations.

9.5 Implementation

Implementation 9.21 (Conversion functions) *Since they are quite complicated, we will not describe the routines for creating adaptive cluster bases and recompressing $\mathcal{H}$- or $\mathcal{H}^2$-matrices in detail. The following functions are collected in the module* `h2conversion`*:*

```
pclusterbasis
buildrow2_supermatrix(psupermatrix s, pccluster root,
                      double eps, int kmax,
                      TruncationStrategy strategy);

pclusterbasis
buildcol2_supermatrix(psupermatrix s, pccluster root,
                      double eps, int kmax,
                      TruncationStrategy strategy);
```

These functions create an adaptive orthogonal nested cluster basis for a given `supermatrix` *and a given cluster tree* `root`*. The parameter* `strategy` *controls the truncation strategy and can take the following values:*

- `HLIB_FROBENIUS_ABSOLUTE` *means that the rank k will be chosen large enough to ensure that the absolute Frobenius error of the resulting matrix is bounded by* `eps`*.*
- `HLIB_FROBENIUS_RELATIVE` *means that the rank k will be chosen large enough to ensure that the blockwise relative Frobenius error of the resulting matrix is bounded by* `eps`*.*
- `HLIB_EUCLIDEAN_RELATIVE` *means that the rank k will be chosen large enough to ensure that the blockwise relative Euclidean error of the resulting matrix is bounded by* `eps`*.*

The parameter `kmax` *gives an absolute upper bound for the rank k.*

As soon as we have orthogonal row and column cluster bases, we can use the function

```
psupermatrix
project2bases_supermatrix(psupermatrix s, int mixed,
                          pclusterbasis row, pclusterbasis col);
```

to create an $\mathcal{H}^2$-matrix that is the optimal approximation of `s` *in the same block structure and with the bases* `row` *and* `col` *(cf. Lemma 9.18). If the parameter* `mixed` *is non-zero, the routine will not create a "pure" $\mathcal{H}^2$-matrix but mix* `rkmatrix` *and* `uniformmatrix` *blocks to minimize the storage complexity.*

Of course there are also top-level functions that hide all details from the user:

```
psupermatrix
buildh2_supermatrix(psupermatrix s, pccluster row, pccluster col,
                    double eps, int kmax,
                    TruncationStrategy strategy);

psupermatrix
buildh2symm_supermatrix(psupermatrix s, pccluster ct,
                        double eps, int kmax,
                        TruncationStrategy strategy);
```

The first routine computes an $\mathcal{H}^2$-matrix that approximates the given matrix `s`*. Here,* `row` *and* `col` *give the row and column cluster trees and* `eps`*,* `kmax` *and* `strategy` *have the same meaning as in the cluster bases construction routines above.*

The second routine is meant for symmetric matrices. Here, we can use the same cluster basis for row and columns, so we can reduce the computational and storage complexity.

Chapter 10

Matrix Equations

In this chapter we consider matrix equations of the form

$$\text{(Lyapunov)} \qquad AX + XA^T + C = 0, \tag{10.1}$$

$$\text{(Sylvester)} \qquad AX - XB + C = 0, \tag{10.2}$$

$$\text{(Riccati)} \qquad AX + XA^T - XFX + C = 0, \tag{10.3}$$

where A, B, C, F are given matrices of suitable dimension and X is the sought solution (a matrix). In the previous chapters we had the task to assemble a (BEM) stiffness matrix A, or to invert the matrix A in a data-sparse format, or to solve an equation $Ax = b$ for the vector x, where the right-hand side b and system matrix A are given. The solution x was just a standard vector.

For matrix equations, also the solution matrix X has to be sought in a data-sparse format. Here, we will seek X either in the $R(k)$-matrix or $\mathcal{H}$-matrix format.

10.1 Motivation

Let us start with a simple example, namely the equation

$$AX + XA + I = 0, \qquad A \in \mathbb{R}^{n\times n}.$$

The solution is the matrix $X = -\frac{1}{2}A^{-1}$, i.e., the inversion of matrices is a special case of such a (linear) matrix equation. From Section 5 we know that the inverse of an elliptic operator can be approximated efficiently in the $\mathcal{H}$-matrix format. In the following we will prove that this can be generalised and we will present some algorithms by which the solution can be computed efficiently in the $R(k)$-matrix or $\mathcal{H}$-matrix format.

10.2 Existence of Low Rank Solutions

The existence of (approximate) solutions in a special format is best answered for the Sylvester equation (which covers the Lyapunov case). For the non-linear Riccati equation we can derive the results easily for low rank F.

For the existence and uniqueness of solutions to the Sylvester equation (10.2) it is necessary and sufficient that $\sigma(A) \cap \sigma(B) = \emptyset$. We demand a slightly stronger condition, namely

$$\sigma(A) < \sigma(B), \tag{10.4}$$

which can be generalised to the case that the spectra of A and B are contained in two disjoint convex sets. Then the solution X is explicitly known.

Theorem 10.1 *Let $A \in \mathbb{R}^{n\times n}, B \in \mathbb{R}^{m\times m}$ and let (10.4) hold. Then the matrix*

$$X := \int_0^\infty \exp(tA)C\exp(-tB)\,\mathrm{d}t \tag{10.5}$$

solves (10.2).

Proof: Due to (10.4) the function $t \mapsto \|\exp(tA)\|\|\exp(-tB)\|$ decays exponentially so that the matrix X is well defined. Now we can simply calculate

$$\begin{aligned} AX - XB &= \int_0^\infty (A\exp(tA)C\exp(-tB) - \exp(tA)C\exp(-tB)B)\ \mathrm{d}t \\ &= \int_0^\infty \frac{\partial}{\partial_t}\exp(tA)C\exp(-tB)\,\mathrm{d}t \\ &= \lim_{t\to\infty} \exp(tA)C\exp(-tB) - \exp(0\cdot A)C\exp(-0\cdot B) \\ &= -C. \end{aligned}$$

■

Since the integrand in (10.5) decays exponentially, we can use special quadrature formulae that need only $\mathcal{O}(\log(\varepsilon)^2)$ quadrature points in order to approximate X up to an absolute error of ε. These can be derived by piecewise interpolation, where the pieces are refined towards the origin, or one can use the formula from the book [58] as done in [34] that gives k weights w_j and points t_j so that the matrix

$$X_k := \sum_{j=1}^k w_j \exp(t_j A)C\exp(-t_j B) \tag{10.6}$$

fulfils the error estimate

$$\|X - X_k\|_2 \le C_{A,B}\|C\|_2\exp(-\sqrt{k}),$$

where the constant $C_{A,B}$ depends on the location of the spectra of A and B. If C is an $R(k_C)$-matrix, then X_k is an $R(k\cdot k_C)$-matrix, which proves the existence of low rank solutions to the Sylvester and Lyapunov equation if the matrix C is of low rank. For the solution X of the Riccati equation there holds

$$AX + XA^T + (C - XFX) = 0,$$

so that the conclusions from above show that the solution X can be approximated up to an error of $C_{A,A^T}\|C - XFX\|_2\exp(-\sqrt{k})$ by a matrix X_k of rank at most $k\cdot(k_C + k_F)$.

The computation of an approximate solution by formula (10.6) is not straight-forward, since it involves the matrix exponential.

10.3 Existence of $\mathcal{H}$-matrix Solutions

Formula (10.6) can be used to prove the existence of $\mathcal{H}$-matrix solutions to the Sylvester equation for $\mathcal{H}$-matrices $C \in \mathcal{H}(T, k_C)$, if both $\exp(t_j A)$ and $\exp(-t_j B)$ can be approximated by $\mathcal{H}$-matrices $A_j \in \mathcal{H}(T, k_A), B_j \in \mathcal{H}(T, k_B)$ for all t_j: Lemma 7.19 proves

$$A_j C B_j \in \mathcal{H}(T,\ C_{\mathrm{id}}^2 C_{\mathrm{sp}}^2 (p+1)^2 \max\{k_A, k_B, k_C\})$$

so that the approximant

$$X_{\mathcal{H},k} := \sum_{j=1}^k w_j A_j C B_j$$

is contained in $\mathcal{H}(T, k_X)$ for a rank $k_X = \mathcal{O}(kp^2 \max\{k_A, k_B, k_C\})$. The existence of $\mathcal{H}$-matrix approximations to the matrix exponential is studied in [22, 34] and briefly in the outlook in chapter 11.3.2. Essentially, one requires the uniform approximability of $(\lambda I - A)^{-1}$ in $\mathcal{H}(T, k_A)$ for complex numbers λ outside the spectrum of A. This is just a variant of the approximation result from Section 5, if the matrix A stems from the discretisation of a partial differential operator of elliptic type.

For a simple $\mathcal{H}$-matrix format, we can improve the analytic results from above by algebraic arguments. Let us consider a block-system of the form

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{bmatrix} - \begin{bmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} + \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = 0,$$

where the off-diagonal blocks $A_{ij}, B_{ij}C_{ij}$, $i \neq j$, are of the $R(k)$-matrix format.

For the two off-diagonal blocks we get the equations

$$A_{22}X_{21} - X_{21}B_{11} + (C_{21} + A_{21}X_{11} - X_{22}B_{21}) = 0,$$

which is a Sylvester equation with low rank matrix $C_{21} + A_{21}X_{11} - X_{22}B_{21}$, so that here the results from the previous section prove that X_{21} can be approximated in the $R(k)$-matrix format.

In the diagonal subblocks this resolves into two smaller Sylvester equations

$$A_{ii}X_{ii} - X_{ii}B_{ii} + (C_{ii} + A_{ij}X_{ji} - X_{ij}B_{ji}) = 0, \quad i \neq j.$$

Since A_{ij} and B_{ji} are of low rank, the (blockwise) rank of C_{11} is increased only by $2k$.

By induction this proves

$$X \approx X_{\mathcal{H},\varepsilon} \in \mathcal{H}(T, \tilde{k}), \quad \|X - X_{\mathcal{H},\varepsilon}\|_2 \leq \varepsilon$$

where T is a block cluster tree where all off-diagonal blocks are leaves (admissible), $A, B, C \in \mathcal{H}(T, k)$ and $\tilde{k} = \mathcal{O}(\log(\varepsilon)^2 kp)$.

10.4 Computing the solutions

The computation of `rkmatrix` and $\mathcal{H}$-matrix solutions can be done efficiently by numerous methods. Here, we want to present only some concepts of this rapidly developing field of research.

Via the matrix exponential

The first method is just the straight-forward implementation of (10.6). We compute the weights w_j and quadrature points t_j beforehand (by an explicit formula), use the $\mathcal{H}$-matrix-by-matrix multiplication $\odot$ to compute $\exp(t_j A)C\exp(-t_j B)$ and need to compute "only" the matrix exponential for all t_j. This can be done by scaling and squaring in the formatted $\mathcal{H}$-matrix arithmetic [14] or by a Dunford-Cauchy representation that involves the resolvents [22]. For each quadrature point we have to compute at least once a matrix exponential, i.e., the complexity is at least $\mathcal{O}(n \log^4 n)$ for this ansatz — neglecting the rank k. The advantage of this method is its simplicity: all computations can be performed in the standard $\mathcal{H}$-matrix arithmetic and it works for matrices A, B, C in $\mathcal{H}$-matrix or `rkmatrix` format.

Via multigrid methods

Multigrid methods rely on a multilevel discretisation of the underlying continuous equation, i.e., we have a sequence

$$A_\ell X_\ell - X_\ell B_\ell + C_\ell = 0$$

of equations where the size of the system on level ℓ is $n_\ell \times n_\ell$ and

$$1 \approx n_1 \leq \cdots \leq n_\ell.$$

Each of the operators A_ℓ, B_ℓ, C_ℓ is discretised on a grid τ_ℓ (e.g., by finite elements) and between two subsequent grids $\tau_\ell, \tau_{\ell+1}$ we have transfer operators P_ℓ that map a vector on the coarser grid τ_ℓ to the finer grid $\tau_{\ell+1}$. Therefore, the coarse solutions X_ℓ can be approximated by

$$X_\ell \approx P_\ell^T X_{\ell+1} P_\ell.$$

In the low rank setting the application of P reads

$$P_\ell^T X_{\ell+1} P_\ell = \sum_{\nu=1}^{k} P_\ell^T a_\nu b_\nu^T P_\ell = \sum_{\nu=1}^{k} (P_\ell^T a_\nu)(P_\ell^T b_\nu)^T,$$

so that the transfer of the solution matrix X between the grids is just the application of the transfer operator to the vectors a_ν, b_ν in the $R(k)$-matrix format. Similarly, this can be done for $\mathcal{H}$-matrix solutions X_ℓ. Apart from the grid transfer operators we need on each level ℓ a so-called smoothing operator S_ℓ, e.g., Richardson or Jacobi [39]. For a Sylvester equation the Jacobi smoother requires the solution of a diagonal Sylvester equation (A and B diagonal matrices).

Via meta methods

A meta method is, e.g., *ADI*. Here, the Sylvester equation is split into a series of linear systems

$$\tilde{A}_{\ell,j} X_{\ell,j} = \tilde{C}_{\ell,j}$$

where $\tilde{A}_{\ell,j} = A_\ell + \lambda_j I$. For the right choice of shift parameters λ_j the iteration converges rapidly, but in each step we have to invert the shifted system $\tilde{A}_{\ell,j}$ — similar to the matrix exponential ansatz in the Dunford-Cauchy representation. In the low rank setting however, one can use standard solvers for the linear system $\tilde{A}_{\ell,j} x = c$, hence the name "meta method".

Via the matrix sign function

The matrix sign function has proven to be a useful tool for the solution of dense matrix equations [52]. Therefore, it is natural to use the same method but formulate it in terms of $\mathcal{H}$-matrices and $\mathcal{H}$-matrix arithmetics. We will explain this for the Lyapunov equation and leave the Riccati equation to the interested reader [34]. We define the matrices

$$X_0 := C, \qquad A_0 := A$$

where $\sigma(A) \subset \mathbb{C}_-$ is assumed. The solution X is just $X = \frac{1}{2} X_\infty$, where X_∞ is the limit of the series

$$X_{i+1} := \frac{1}{2}(X_i + A_i^{-T} X_i A_i^{-1}), \quad A_{i+1} := \frac{1}{2}(A_i + A_i^{-1}).$$

Implicitly, we have computed

$$\operatorname{sign} \begin{bmatrix} A^T & C \\ & -A \end{bmatrix} = \begin{bmatrix} A_\infty^T & X_\infty \\ & -A_\infty \end{bmatrix}$$

by Newton's method (sign is the matrix sign function corresponding to the complex sign function on $\mathbb{C} \setminus \{0\}$:

$$\operatorname{sign}(r + ic) = \begin{cases} 1 & r > 0 \\ -1 & r < 0 \end{cases}$$

Locally, the convergence is quadratic but the initial slow (linear) convergence dominates. Therefore, one should use an acceleration technique by scaling.

We consider the scalar case $a_0 \ll 0$ and compute $a_{i+1} := \frac{1}{2}(a_i + a_i^{-1}) \approx \frac{1}{2} a_i$. The limit is the same if we scale the iterate a_0 by some value $\alpha > 0$:

$$\operatorname{sign}(a_0) = \lim_{i \to \infty} a_i = \lim_{i \to \infty} b_i = \operatorname{sign}(b_0), \qquad b_0 = \alpha a_0, \quad b_{i+1} := \frac{1}{2}(b_i + b_i^{-1}).$$

In the limit we get $\lim_{i\to\infty} a_i = -1$, but for $a_0 \ll 0$ we have just a linear convergence rate of $1/2$. Here, it is obvious to compute

$$b_0 := a_0/|a_0|,$$

i.e., we rescale the iterate a_0 and get the solution in one step. For matrices A_i this translates to

$$\begin{aligned} \alpha &:= \sqrt{\|A^{-1}\|_2/\|A\|_2}, \\ A_0 &:= \alpha A, \\ A_{i+1} &= \frac{1}{2}(A_i + A_i^{-1}). \end{aligned}$$

In principle, one can use the acceleration by scaling in every step of Newton's method, but for two reasons this is not adavantageous: first, the quadratic convergence can be deteriorated and second, the truncation error can be amplified by the scaling factor. Therefore, one should only use the scaling in the first step to balance the spectrum of A_0. Due to the fact that we have to compute A_0^{-1} anyway, the additional cost to compute α by some power iteration (`norm2_supermatrix`) is negligible.

In $\mathcal{H}$-matrix arithmetics we have to replace the exact addition, multiplication and inversion by the formatted counterparts. This yields a fast solution method for Lyapunov, Sylvester and Riccati equations where the matrices A, B, C, F are allowed to be of an arbitrary $\mathcal{H}$-matrix format. Since the computation of the matrix sign involves $\log(n)$ inversions, the overall complexity of this approach is $\mathcal{O}(n \log^3 n)$.

10.5 High-dimensional Problems

As a generalisation to the Sylvester equation of the previous section one can consider equations $Ax = b$ where the matrix $A \in \mathbb{R}^{n^d \times n^d}$ is of the form

$$A = \sum_{i=1}^{d} \widehat{A}_i, \qquad \widehat{A}_i = \underbrace{I \otimes \cdots \otimes I}_{i-1 \text{ terms}} \otimes A_i \otimes \underbrace{I \otimes \cdots \otimes I}_{d-i \text{ terms}}, \qquad I, A_i \in \mathbb{R}^{n \times n}$$

and the right-hand side is given as a tensor vector

$$b = \bigotimes_{i=1}^{d} b_i, \qquad b_i \in \mathbb{R}^n, \quad b[j_1, \ldots, j_d] = \prod_{i=1}^{d} b_i[j_i] \quad \text{for } j \in \{1, \ldots, n\}^d.$$

Then the solution can be computed by use of the matrix exponential $\exp(t_j A_i)$ of the $n \times n$ matrices A_i,

$$x \approx \sum_{\nu=1}^{k} \bigotimes_{i=1}^{d} x_{i,\nu}, \qquad x_{i,\nu} = w_\nu \exp(t_\nu A_i) b_i,$$

such that the complexity is reduced to $\mathcal{O}(dn \log(n)^c)$ instead of $\mathcal{O}(n^d)$ for the `fullmatrix` solution. For details the reader is referred to [28]. Here, the matrix exponential $\exp(t_\nu A_i)$ can be approximated in the $\mathcal{H}$-matrix format and the solution is represented as a (low Kronecker rank) tensor vector, which is a d-dimensional analogue to the `rkmatrix` format.

Chapter 11

Outlook

In the previous chapters we introduced the hierarchical matrix format ($\mathcal{H}$-matrices) and a refined variant, the $\mathcal{H}^2$-matrix format. In this chapter we will consider further algorithms for these matrices, other (similar) matrix formats and possible applications.

11.1 Adaptive Refinement

For the efficient treatment of boundary integral and partial differential equations an adaptive discretization scheme is inevitable, that means for a given right-hand side the unique solution has to be approximated on a grid with as few as necessary unknowns. This is typically done by refining the grid adaptively according to the indication by some a posteriori error estimator.

For partial differential equations the discretization itself is of negligible complexity, only the solution of the typically ill-conditioned large system of equations is a task. If a large part of the grid is refined, then the (block) cluster tree is best constructed in the standard geometrically or cardinality balanced way and the formatted $\mathcal{H}$-matrix inversion yields a good approximate inverse that can be used to directly solve or precondition the system. If only a small part of the grid is to be refined, then this can be regarded as a low rank perturbation of the operator on the coarse grid. The Sherman-Morrison-Woodbury formula for the inversion of the perturbed matrix allows for an efficient low rank update of the previously computed inverse for the coarse grid.

The story is different for (boundary) integral operators. Here, the (data-sparse) discretization is of high complexity. Therefore, it is desirable to retain as many as possible entries when refining the grid. This can be achieved for the regular geometrically balanced clustering if one uses the approximation by interpolation from Chapter 3 or the cross approximation techniques from Chapter 4. This topic is partially covered by [30] and [31] and it is the topic of the doctoral thesis of Jelena Djokić.

11.2 Other Hierarchical Matrix Formats

The hierarchical matrices introduced in the previous chapters consist of `rkmatrix` or `fullmatrix` blocks (`uniformmatrix` blocks for $\mathcal{H}^2$-matrices). One may think of other formats like Toeplitz matrices, banded matrices, sparse matrices or others. Here, one has to distinguish between the data-sparse storage and fast evaluation of the matrix, and the possibility to apply the formatted $\mathcal{H}$-matrix arithmetics. While the first goal, the storage and evaluation, can be achieved for a variety of formats, this is not true for the second task, the formatted arithmetics. Toeplitz matrices allow for a fast inversion, but as subblocks in a hierarchical matrix they have to be added and multiplied by other matrix blocks, which will destroy the structure.

11.3 Applications of Hierarchical Matrices

11.3.1 Partial Differential Equations

$\mathcal{H}^2$-matrices and multi-grid methods share many properties in common, so it is straightforward to look for approximations of the inverses of elliptic partial differential equations in the $\mathcal{H}^2$-matrix format.

A simple method to compute this approximation is to use the $\mathcal{H}$-matrix inversion and use the algorithm from [9] to find suitable cluster bases. Evaluating the resulting $\mathcal{H}^2$-representation of the inverse is more efficient than evaluating the $\mathcal{H}$-matrix representation, and the complexity of the transformation from $\mathcal{H}$- to $\mathcal{H}^2$-format is lower than that of the $\mathcal{H}$-inversion.

Finding an algorithm that allows us to create an $\mathcal{H}^2$-matrix inverse without having to resort to the $\mathcal{H}$-matrix inversion is a topic of current research, cf. [12, 11].

11.3.2 Matrix Functions

The most prominent matrix function is $M \mapsto M^{-1}$, i.e., $f(x) = \frac{1}{x}$, which we have already analysed in Chapter 5 for elliptic partial differential operators. Another important function is the matrix exponential, because it allows us to solve systems of ordinary differential equations:

$$\dot{x}(t) = Ax(t), \quad x(0) = x_0 \qquad \Rightarrow \qquad x(t) = \exp(tA)x_0.$$

If we could compute $\exp(\delta A)$ for a small $\delta > 0$, then we can easily obtain the values of x at all times $t_j = j\delta$, $j = 1, \ldots, N$, by j times evaluating $\exp(\delta A)$: $x(t_j) = (\exp(\delta A))^j x_0$, i.e., we need N times the matrix-vector multiplication and only once the computation of the matrix exponential.

The question remains whether or not it is possible to approximate the matrix exponential in the $\mathcal{H}$-matrix format and how one can efficiently compute the matrix exponential in this format. The first question is answered in [22] under the assumption that the resolvents $(A - \lambda I)^{-1}$ can be approximated in the $\mathcal{H}$-matrix format, cf. Chapter 5. For the second question there is no definitive answer; often the standard scaling and squaring strategy proposed in [14] does the job very well but for a more general survey the reader is referred to [50].

For other matrix functions ([23],[24]) one can use the representation by the Cauchy integral

$$f(M) = \frac{1}{2\pi i} \oint_\Gamma f(t)(M - tI)^{-1} \mathrm{d}t$$

and an efficient (exponentially convergent) quadrature rule

$$f(M) \approx \sum_{j=1}^{k} w_j f(t_j)(M - t_j I)^{-1}$$

to obtain an approximation.

Index

Bibliography

[1] M. Bebendorf. Approximation of boundary element matrices. *Numer. Math.*, 86(4):565–589, 2000.

[2] M. Bebendorf. *Effiziente numerische Lösung von Randintegralgleichungen unter Verwendung von Niedrigrang-Matrizen.* PhD thesis, Universität Saarbrücken, 2000.

[3] M. Bebendorf and W. Hackbusch. Existence of $\mathcal{H}$-matrix approximants to the inverse FE-matrix of elliptic operators with L^∞-coefficients. *Numerische Mathematik*, 95:1–28, 2003.

[4] M. Bebendorf and S. Rjasanow. Adaptive Low-Rank Approximation of Collocation Matrices. *Computing*, 70(1):1–24, 2003.

[5] Mario Bebendorf. Efficient inversion of the Galerkin matrix of general second order elliptic operators with non-smooth coefficients. Technical Report 6, Max Planck Institute for Mathematics in the Sciences, 2004.

[6] Mario Bebendorf. On the numerical solution of convection-dominated problems using hierarchical matrices. Technical Report 78, Max Planck Institute for Mathematics in the Sciences, 2005.

[7] S. Börm. $\mathcal{H}^2$-matrices — multilevel methods for the approximation of integral operators. *Comput. Visual. Sci.*, 7:173–181, 2004.

[8] S. Börm and L. Grasedyck. Low-rank approximation of integral operators by interpolation. *Computing*, 72:325–332, 2004.

[9] S. Börm and W. Hackbusch. Data-sparse approximation by adaptive $\mathcal{H}^2$-matrices. *Computing*, 69:1–35, 2002.

[10] S. Börm and W. Hackbusch. $\mathcal{H}^2$-matrix approximation of integral operators by interpolation. *Applied Numerical Mathematics*, 43:129–143, 2002.

[11] Steffen Börm. Adaptive variable-rank approximation of dense matrices. Preprint 114/2005, Max Planck Institute for Mathematics in the Sciences, 2005. Submitted to SIAM Journal of Matrix Analysis and Applications.

[12] Steffen Börm. $\mathcal{H}^2$-matrix arithmetics in linear complexity. *Computing*, 77(1):1–28, 2006.

[13] Steffen Börm and Lars Grasedyck. Hybrid cross approximation of integral operators. *Numerische Mathematik*, 101:221–249, 2005.

[14] Steffen Börm, Lars Grasedyck, and Wolfgang Hackbusch. An introduction to hierarchical matrices. *Mathematica Bohemica*, 127(2):229–241, 2002.

[15] Steffen Börm and Wolfgang Hackbusch. Approximation of boundary element operators by adaptive $\mathcal{H}^2$-matrices. *Foundations of Computational Mathematics*, 312:58–75, 2004.

[16] Steffen Börm, Maike Löhndorf, and Jens Markus Melenk. Approximation of integral operators by variable-order interpolation. *Numerische Mathematik*, 99(4):605–643, 2005.

[17] A. Brandt, S. McCormick, and J. Ruge. Algebraic multigrid (AMG) for sparse matrix equations. In *Sparsity and its Applications*, pages 257–284. Cambridge University Press, 1984.

[18] O. Bröker, M. Grote, C. Mayer, and A. Reusken. Robust parallel smoothing for multigrid via sparse approximate inverses. *SIAM J. Sci. Comput.*, 23:1396–1417, 2001.

[19] R. A. DeVore and G. G. Lorentz. *Constructive Approximation*. Springer-Verlag, 1993.

[20] Georg Dolzmann and Stefan Müller. Estimates for green's matrices of elliptic systems by L^p theory. *Manuscripta Math.*, pages 261–273, 1995.

[21] C. F. Van Loan G. H. Golub. *Matrix Computations*. Johns Hopkins University Press, London, 1996.

[22] I. Gavrilyuk, W. Hackbusch, and B. Khoromskij. $\mathcal{H}$-matrix approximation for the operator exponential with applications. *Numerische Mathematik*, 92:83–111, 2002.

[23] Ivan Gavrilyuk, Wolfgang Hackbusch, and Boris Khoromskij. Data-sparse approximation to operator-valued functions of elliptic operator. *Mathematics of Computation*, 73:1107–1138, 2004.

[24] Ivan Gavrilyuk, Wolfgang Hackbusch, and Boris Khoromskij. Data-sparse approximation of a class of operator-valued functions. *Mathematics of Computation*, 74:681–708, 2005.

[25] K. Giebermann. Multilevel approximation of boundary integral operators. *Computing*, 67:183–207, 2001.

[26] S. A. Goreinov, E. E. Tyrtyshnikov, and N. L. Zamarashkin. A theory of pseudoskeleton approximations. *Lin. Alg. Appl.*, 261:1–22, 1997.

[27] L. Grasedyck. *Theorie und Anwendungen Hierarchischer Matrizen*. PhD thesis, Universität Kiel, 2001.

[28] L. Grasedyck. Existence and computation of low Kronecker-rank approximations for large linear systems of tensor product structure. *Computing*, 72:247–265, 2004.

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

[30] L. Grasedyck, W. Hackbusch, and S. LeBorne. Adaptive refinement and clustering of $\mathcal{H}$-matrices. Preprint 106/2001, Max Planck Institute of Mathematics in the Sciences, 2001.

[31] L. Grasedyck, W. Hackbusch, and S. LeBorne. Adaptive geometrically balanced clustering of $\langle$-matrices. *Computing*, 73:1–23, 2004.

[32] L. Grasedyck, R. Kriemann, and S. LeBorne. Domain-decomposition based $\mathcal{H}$-matrix preconditioners. In *Proceedings of DD16*, LNSCE. Springer-Verlag Berlin. to appear.

[33] L. Grasedyck and S. LeBorne. H-matrix preconditioners in convection-dominated problems. *SIAM J. Mat. Anal.*, 2005. to appear.

[34] Lars Grasedyck, Wolfgang Hackbusch, and Boris Khoromskij. Solution of large scale algebraic matrix Riccati equations by use of hierarchical matrices. *Computing*, 70:121–165, 2003.

[35] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *Journal of Computational Physics*, 73:325–348, 1987.

[36] M. Güter and K.-O. Widman. The green function for uniformly elliptic equations. *Manuscripta Mathematica*, 37:303–342, 1982.

[37] G. Haase, M. Kuhn, and S. Reitzinger. Parallel AMG on distributed memory computers. *SIAM J. Sci. Comp.*, 24(2):410–427, 2002.

[38] W. Hackbusch. *Elliptic Differential Equations. Theory and Numerical Treatment*. Springer-Verlag Berlin, 1992.

[39] W. Hackbusch. *Iterative Solution of Large Sparse Systems.* Springer-Verlag New York, 1994.

[40] W. Hackbusch. A sparse matrix arithmetic based on $\mathcal{H}$-matrices. Part I: Introduction to $\mathcal{H}$-matrices. *Computing*, 62:89–108, 1999.

[41] W. Hackbusch. Direct domain decomposition using the hierarchical matrix technique. In I. Herrera, D. Keyes, O. Widlund, and R. Yates, editors, *Domain Decomposition Methods in Science and Engineering*, pages 39–50. UNAM, 2003.

[42] W. Hackbusch and B. Khoromskij. A sparse $\mathcal{H}$-matrix arithmetic: General complexity estimates. *J. Comp. Appl. Math.*, 125:479–501, 2000.

[43] W. Hackbusch, B. Khoromskij, and R. Kriemann. Hierarchical matrices based on a weak admissibility criterion. *Computing*, 73:207–243, 2004.

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

[45] Wolfgang Hackbusch, Boris Khoromskij, and Stefan Sauter. On $\mathcal{H}^2$-matrices. In H. Bungartz, R. Hoppe, and C. Zenger, editors, *Lectures on Applied Mathematics*, pages 9–29. Springer-Verlag, Berlin, 2000.

[46] R. Kriemann. Parallel H-matrix arithmetics on shared memory systems. *Computing*, 74:273–297, 2005.

[47] C. Lage, G. Schmidlin, and C. Schwab. Rapid solution of first kind boundary integral equations in $\mathbb{R}^3$. *Engineering Analysis with Boundary Elements*, 27:469–490, 2003.

[48] M. Lintner. The eigenvalue problem for the 2d Laplacian in $\mathcal{H}$-matrix arithmetic and application to the heat and wave equation. *Computing*, 72:293–323, 2004.

[49] G. Meinardus. *Approximation of Functions: Theory and Numerical Methods.* Springer-Verlag New York, 1967.

[50] C. Moler and C. F. Van Loan. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM Reviews*, 45:3–49, 2003.

[51] T.J. Rivlin. *The Chebyshev Polynomials.* Wiley-Interscience, New York, 1984.

[52] J. D. Roberts. Linear model reduction and solution of the algebraic Riccati equation by use of the sign function. *Internat. J. Control*, 32:677–687, 1980.

[53] V. Rokhlin. Rapid solution of integral equations of classical potential theory. *Journal of Computational Physics*, 60:187–207, 1985.

[54] R. W. Ruge and K. Stüben. Efficient solution of finite difference and finite element equations by algebraic multigrid (AMG). In H. Holstein D. J. Paddon, editor, *Multigrid Methods for Integral and Differential Equations*, pages 169–212. Clarenden Press Oxford, 1985.

[55] S. Sauter. Variable order panel clustering (extended version). Preprint 52/1999, Max-Planck-Institut für Mathematik, Leipzig, Germany, 1999.

[56] S. Sauter. Variable order panel clustering. *Computing*, 64:223–261, 2000.

[57] Robert Schreittmiller. *Zur Approximation der Lösungen elliptischer Systeme partieller Differentialgleichungen mittels Finiter Elemente und $\mathcal{H}$-Matrizen.* PhD thesis, TU München, 2005.

[58] F. Stenger. *Numerical methods based on Sinc and analytic functions.* Springer, New York, 1993.