Hierarchical Matrices

Steffen Börm Lars Grasedyck Wolfgang Hackbusch

April 21, 2005

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 $CN^{\sigma}_{t+\Delta t} = C2^{\sigma}N^{\sigma}_t$ 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: $f(D)=\operatorname{diag}\{f(d_i):1\le i\le N\}$.

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 $\mathcal{F}=\left(\omega^{i+j}\right)_{i,j=1,\dots,N}$ 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., $\#\{(i,j):A_{ij}\ne 0\}=\mathcal{O}(N)$. 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 $\mathcal{O}(\omega^2N)$ 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 (BEM)

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 [23], 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.*

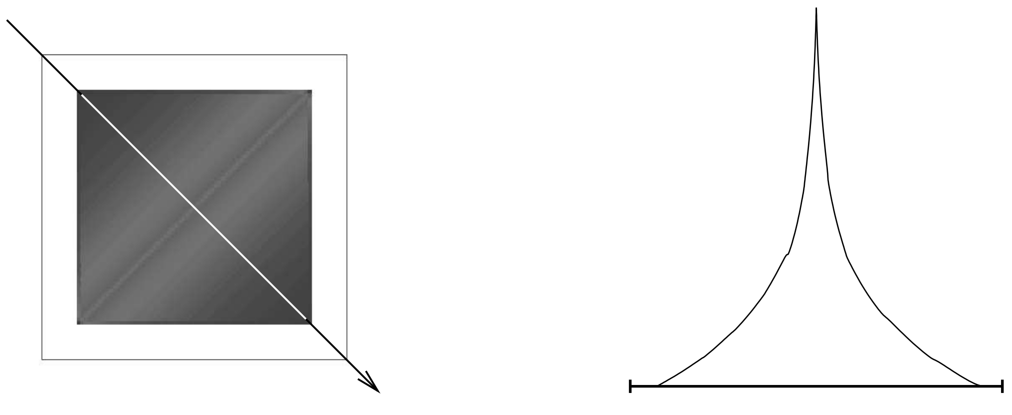


Figure 1.1: The function $\log|x-y|$ in $[0,1]\times[0,1]$ and a cut along the line $x+y=1$.

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

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

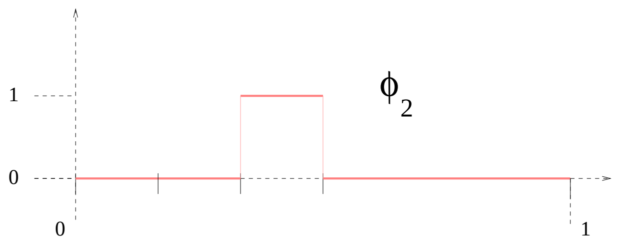
and seek the discrete solution $\mathcal{U}_n$ in the same space V_n, i.e., $\mathcal{U}_n = \sum_{j=0}^{n-1} u_j\varphi_j$ such that the coefficient vector u is the solution of the linear system

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

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

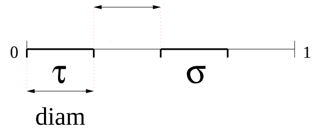
$$g(x,y) = \sum_{\nu=0}^{\infty} \frac{1}{\nu!}\partial_x^\nu g(x_0,y)(x-x_0)^\nu.$$

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 $(i,j) \in t \times s$:

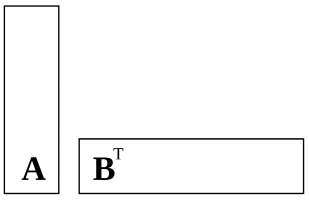
$$\tilde{G}_{ij} := \int_0^1 \int_0^1 \varphi_i(x) \tilde{g}(x,y) \varphi_j(y) \mathrm{d}y \mathrm{d}x.$$

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

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



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 $\tau := \cup_{i\in t} \mathrm{supp} \varphi_i$, $\sigma := \cup_{i \in s} \mathrm{supp}\, \varphi_i$ 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:

$$\begin{aligned}
|G_{ij} - \tilde{G}_{ij}| &= \Big|\int_0^1 \int_0^1 \varphi_i(x)(g(x,y) - \tilde{g}(x,y))\varphi_j(y)\mathrm{d}y\mathrm{d}x\Big| \\
&\overset{(1.5)}{\leq} \int_0^1 \int_0^1 |\varphi_i(x)| 3^{-k} |\varphi_j(y)|\mathrm{d}y\mathrm{d}x \\
&= \frac{3}{2} 3^{-k} \int_0^1 \varphi_i(x)\mathrm{d}x \int_0^1 \varphi_j(y)\mathrm{d}y \\
&= \frac{3}{2} n^{-2} 3^{-k}.
\end{aligned}$$

■

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):

$$\mathcal{I} \times \mathcal{I} = \dot{\bigcup_{\nu=1,\ldots,b}} t_\nu \times s_\nu .$$

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 $t_\nu \times s_\nu$ 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 $T_{\mathcal{I}}$

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.

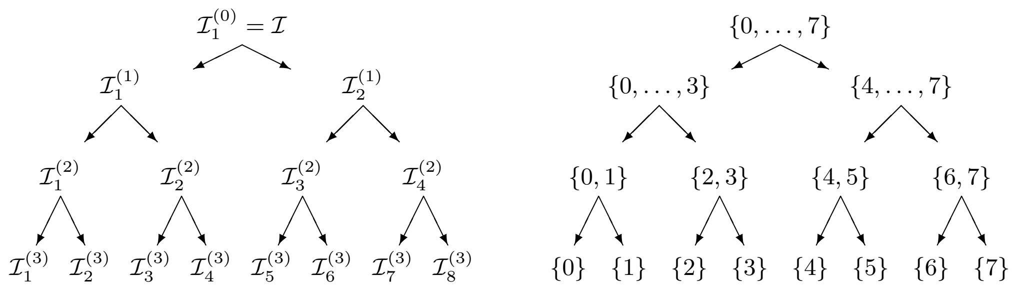


Figure 1.3: The cluster tree $T_\mathcal{I}$ for $p = 3$, on the left abstract and on the right concrete.

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 $\mathcal{I} = \{0,\dots,n-1\}$ 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($\mathcal{I}$,$\mathcal{I}$).

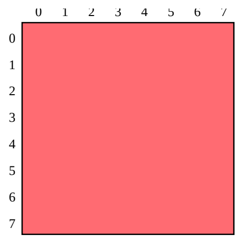
Algorithm 1 Construction of the block cluster tree $T_{\mathcal{I}\times\mathcal{I}}$

```
procedure BuildBlockClusterTree(cluster $t$, $s$)
if $(t,s)$ is admissible then
    $S(t \times s) := \emptyset$
else
    $S(t \times s) := \{t' \times s' \mid t' \in S(t)$ and $s' \in S(s)\}$
    for $t' \in S(t)$ and $s' \in S(s)$ 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*

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

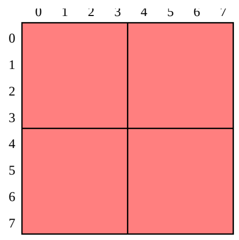


which is not admissible because the corresponding domain to the index set $\{0,\dots,7\}$ *is the interval* $[0,1]$ *and*

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

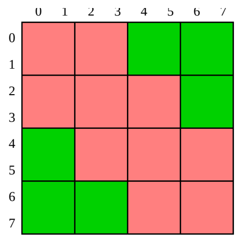
The four successors of the root in the tree $T_{\mathcal{I}\times\mathcal{I}}$ *are*

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



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

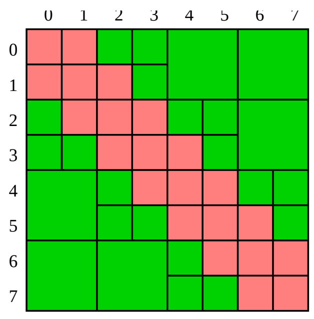
$$\begin{array}{llll}\{0,1\}\times\{0,1\}, & \{0,1\}\times\{2,3\}, & \{0,1\}\times\{4,5\}, & \{0,1\}\times\{6,7\},\\ \{2,3\}\times\{0,1\}, & \{2,3\}\times\{2,3\}, & \{2,3\}\times\{4,5\}, & \{2,3\}\times\{6,7\},\\ \{4,5\}\times\{0,1\}, & \{4,5\}\times\{2,3\}, & \{4,5\}\times\{4,5\}, & \{4,5\}\times\{6,7\},\\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



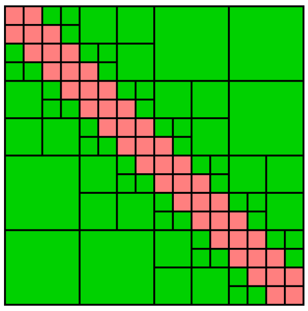
Now some of the nodes are admissible, e.g., the node $\{0,1\}\times\{4,5\}$ *because the corresponding domain is* $[0,\frac{1}{4}]\times[\frac{1}{2},\frac{3}{4}]$*:*

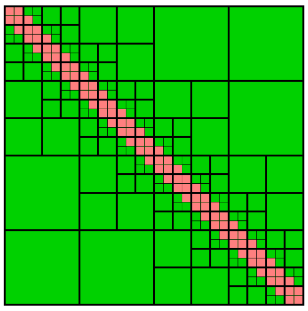
$$\operatorname{diam}\left(\left[0,\frac{1}{4}\right]\right) = \frac{1}{4} = \operatorname{dist}\left(\left[0,\frac{1}{4}\right],\left[\frac{1}{2},\frac{3}{4}\right]\right).$$

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*

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

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 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

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

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

Definition 1.11 (`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 1.12 (`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 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->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.*

1.6.3 Hierarchical Matrix Representation

Definition 1.13 ($\mathcal{H}$-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*

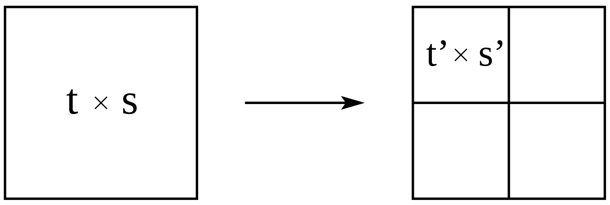
$$\mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, k) := \left\{ M \in \mathbb{R}^{\mathcal{I}\times\mathcal{I}} \mid \operatorname{rank}(M|_{t\times s}) \le k \quad \textit{for all admissible leaves } t \times s \textit{ of } T_{\mathcal{I}\times\mathcal{I}} \right\}.$$

Definition 1.14 ($\mathcal{H}$-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 1.15 (`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` × `block_cols` *submatrices. The size of the matrix is* `rows`×`cols`*, i.e., $M \in \mathbb{R}^{\texttt{rows}\times\texttt{cols}}$. The matrix can be*

- *an* `rkmatrix` *- then* `r` ≠ `0x0`*,* `f` = `0x0` *and* `s` = `0x0`*:*
  the matrix `r` *is the* `rkmatrix` *representation of M;*
- *a* `fullmatrix` *- then* `f` ≠ `0x0`*,* `r` = `0x0` *and* `s` = `0x0`*:*
  the matrix `f` *is the* `fullmatrix` *representation of M;*
- *a* `supermatrix` *- then* `s` ≠ `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

  $$M_{1,1},\ldots,M_{\texttt{blockrows},1},\;M_{1,2},\ldots,M_{\texttt{blockrows},2},\;\;\cdots\;\;,M_{1,\texttt{blockcols}},\ldots,M_{\texttt{blockrows},\texttt{blockcols}}.$$

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 ≡ submatrices) and the submatrices corresponding to admissible or inadmissible leaves are stored in the `rkmatrix` or `fullmatrix` format.

1.7 Exercises

1.7.1 Theory

Exercise 1 (Cardinality of the Cluster Tree $T_{\mathcal{I}}$) *Let $p \in \mathbb{N}_0$ and $n := 2^p$.*

1. *Prove that for $n_{\min} := 1$ the cardinality of the tree $T_{\mathcal{I}}$ is $\#T_{\mathcal{I}} = 2n - 1$.*
2. *What is the cardinality of $T_{\mathcal{I}}$ in the general case $n_{\min} > 1$?*

Exercise 2 (Exact Integration of the Kernel) *The integration in the admissible and inadmissible blocks can be performed analytically (because the domain of integration and the kernel are both simple).*

- *Compute the integrals for the entries $A_{i\nu}, B_{j\nu}$ in the admissible blocks* analytically.

Hint: *split the integration and use $\int \log(x) = x\log(x) - x$*

Exercise 3 (`rkmatrix` Representation) *Let $M \in \mathbb{R}^{n\times m}$ be a matrix and $k \in \mathbb{N}_0$. Prove the following two statements.*

1. *If $M = AB^T$ for matrices $A \in \mathbb{R}^{n\times k}$ and $B \in \mathbb{R}^{m\times k}$ then the rank of M is at most k.*
2. *If the rank of M is at most k then there exist two matrices $A \in \mathbb{R}^{n\times k}$ and $B \in \mathbb{R}^{m\times k}$ such that $M = AB^T$.*

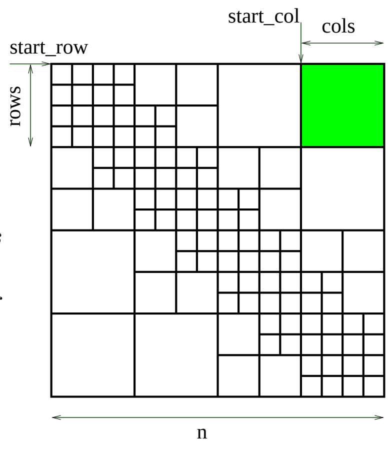
What are the storage requirements (number of `double`*s to be stored) for an $n \times m$ matrix stored in* `rkmatrix` *representation ?*

1.7.2 Practice

Exercise 4 (Assembly of the Matrix Entries) *Write a procedure that allocates and fills a* `rows`$\times$`cols` `rkmatrix` *block with the respective entries from Subsection 1.6.2:*

```
prkmatrix
fill_rkmatrix(int n, int k, int rows, int cols,
              int start_row, int start_col);
```

The number of basis functions is `n`*, the size of the matrix to be allocated is* `rows` $\times$ `cols`*, the indices in this block are* $\{$`start_row` $,\dots,$ `start_row` $+$ `rows` $-1\}$ *for the rows and* $\{$`start_col` $,\dots,$ `start_col` $+$ `cols` $-1\}$ *for the columns.*



Exercise 5 (Matrix-Vector Multiplication) *Implement the matrix-vector multiplication $w := Mv$ for the* `fullmatrix`*,* `rkmatrix` *and* `supermatrix` *format:*

```
void
eval_fullmatrix(pfullmatrix f, double* v, double* w);
void
eval_rkmatrix(prkmatrix r, double* v, double* w);
void
eval_supermatrix(psupermatrix s, double* v, double* w);
```

For an $n \times m$ matrix the `fullmatrix` *version should have a complexity of $\mathcal{O}(nm)$ while this reduces to $\mathcal{O}((n+m)k)$ for the* `rkmatrix` *format. The procedure for the* `supermatrix` *format has to call the matrix-vector multiplication for the submatrices (*`s` $\neq$ `0x0`*), for the* `rkmatrix` *(*`r` $\neq$ `0x0`*) or for the* `fullmatrix` *(*`f` $\neq$ `0x0`*).*

Hint: *It is advisable to first implement the respective* `addeval` *counterparts* $v \mapsto w + Mv$ *and use them to define the* `eval` *functions.*

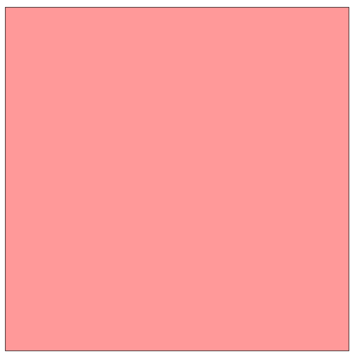
Exercise 6 (Numerical Test) *The* C*-file* `example_1d.c` *contains the main program which invokes the matrix assembly and solves the discrete linear system by the conjugate gradient method (only matrix-vector multiplications are needed) for the two right hand sides* $\mathtt{f}_i$*. The matrix structure is written to "*`matrix.ps`*", the right hand side to "*`rhs.ps`*" and the solution to "*`solution.ps`*". Compile the object files for*

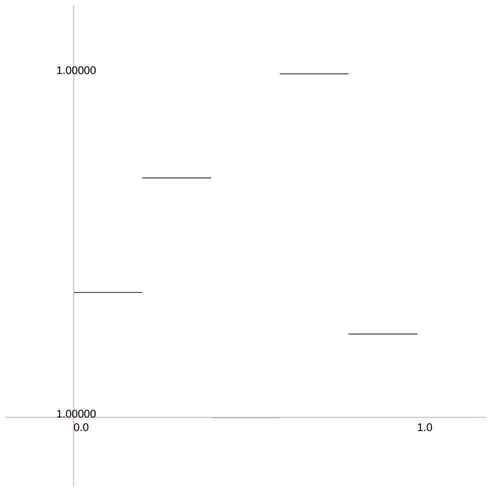
```
fullmatrix.c
rkmatrix.c
supermatrix.c
ie1d.c
solver.c
graphics.c
example_1d.c
```

by "`gcc -c -Wall -pedantic *.c`*" and link the main program by*

```
gcc -o example_1d -Wall -pedantic *.o -lm
```

1. *Invoke the program* `example_1d` *to test your implementation of the* `fullmatrix` *assembly and* `fullmatrix`*-vector multiplication (choose* $n = 5$ *and* $k = 5$ *for the right-hand side number* 1 *such that only one* `fullmatrix` *block appears). The matrix structure (*`matrix.ps`*) and solution (*`solution.ps`*) should look as follows:*

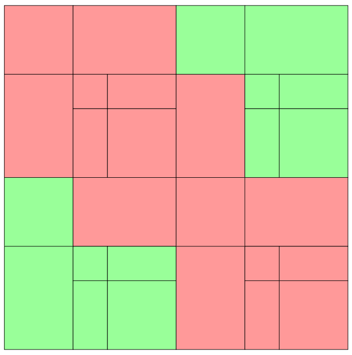


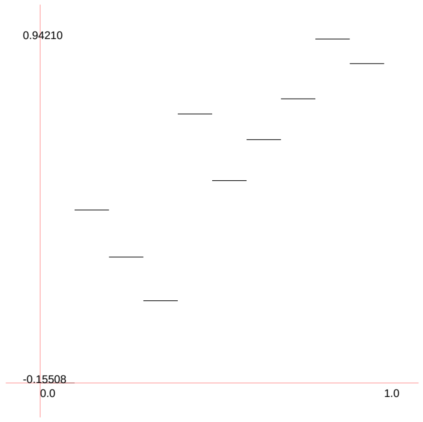


2. *Invoke the program* `example_1d` *to test your implementation of the* `rkmatrix` *assembly. Choose the parameters* $n = 8, k = 1$*, right-hand side* 1*. The output should report*

```
r->a[0] = 0.125
r->b[0] = -0.0470093
r->a[1] = 0.125
r->b[1] = -0.0260784
```

Afterwards, test your implementation of the matrix-vector multiplication (choose $n = 10$ *and* $k = 1$ *for the right-hand side number* 2*). The matrix structure (*`matrix.ps`*) and solution (*`solution.ps`*) should look as follows:*





3. *Look at the (discrete) solutions* $\mathtt{u}_j$ *to the right hand sides* $\mathtt{f}_j$ *for* $j \in \{1, 2\}$ *and larger* n *(100, 200, 500, 1000). Try to guess what the continuous solutions* $\mathcal{U}_j$ *corresponding to the right-hand sides* $\mathcal{F}_j$ *might be.*

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

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 $r = t_0 \neq s$.

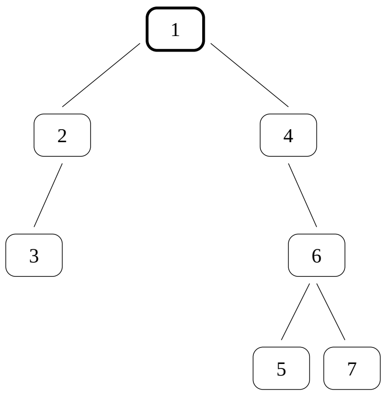
Let $t \in N \setminus \{r\}$, and let $t_0, \ldots, t_m$ be its sequence of ancestors. Due to $t \neq t_0 = r$, we have $m > 0$ and find $t = t_m \in \operatorname{sons}(T, t_{m-1})$. Therefore we can conclude by setting $t^+ := t_{m-1}$. ■

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

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 $\operatorname{level}(T, t)$ if this does not lead to confusion.

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

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



Tree in standard notation

The maximal level is called the *depth* of T and denoted by

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

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 $\operatorname{father}(t) = t^+$. Obviously, we have $\operatorname{level}(t) = 0$ if and only for $t = \operatorname{root}(T)$. A vertex $t \in N$ is a *leaf* if $\operatorname{sons}(t) = \emptyset$ holds and we define the *set of leaves*

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

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:*

- $\widehat{\operatorname{root}(T)} = \mathcal{I}$.
- *For each vertex $t \in N$, we have*

$$s_1, s_2 \in \operatorname{sons}(T, t), s_1 \neq s_2 \Rightarrow \hat{s}_1 \cap \hat{s}_2 = \emptyset \qquad \text{and} \qquad \operatorname{sons}(t) \neq \emptyset \Rightarrow \hat{t} = \bigcup_{s \in \operatorname{sons}(t)} \hat{s}.$$

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 $t \in N$.

The implementation of the tree structure is straightforward:

Implementation 2.6 (`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* $\mathrm{sons}(t)$*, while the array* `son` *is filled with the pointers to these sons.*

Lemma 2.7 *For any cluster tree $T_{\mathcal{I}}$ there holds $\mathcal{I} = \dot{\bigcup}_{v \in \mathcal{L}(T_{\mathcal{I}})} \hat{v}$.*

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 $\Omega_i := \mathrm{supp}(\varphi_i)$. 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.

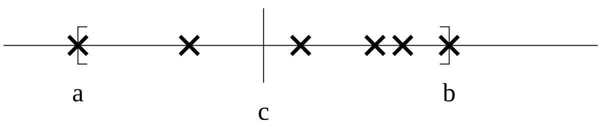
Let us first consider the one-dimensional case: We want to split the index set $\hat{t} \subset \mathcal{I}$ corresponding to a cluster t into two parts. Each index $i \in \hat{t}$ corresponds to a point $x_i \in \mathbb{R}$, so we can set $a := \min\{x_i \ : \ i \in \hat{t}\}$, $b := \max\{x_i \ : \ i \in \hat{t}\}$ 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

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

Obviously, we have

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

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_1} and B_{t_2} by setting $c_j := (a_j + b_j)/2$ and

$$B_{t_1} := [a_1, b_1] \times \cdots \times [a_j, c_j] \times \cdots \times [a_d, b_d] \quad \text{and} \quad B_{t_2} := [a_1, b_1] \times \cdots \times [c_j, b_j] \times \cdots \times [a_d, b_d].$$

The index sets $\hat{t}_1$ and $\hat{t}_2$ are defined by

$$\hat{t}_1 := \{i \in \hat{t} \ : \ x_i \in B_{t_1}\} \quad \text{and} \quad \hat{t}_2 := \mathcal{I} \setminus \hat{t}_2.$$

We set $j_{t_1} := j_{t_2} := (j_t \mod d) + 1$.

Since $x_i \in B_{t_1}$ for $i \in \hat{t}_1$ and $x_i \in B_{t_2}$ for $i \in \hat{t}_2$ hold by construction, we can now repeat the procedure for t_1 and t_2.

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

We will now demonstrate how the second approach can be implemented. 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 $\{0, \ldots, n_{\text{idx}} - 1\}$ 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 2.8 (`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 2.9 (`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.

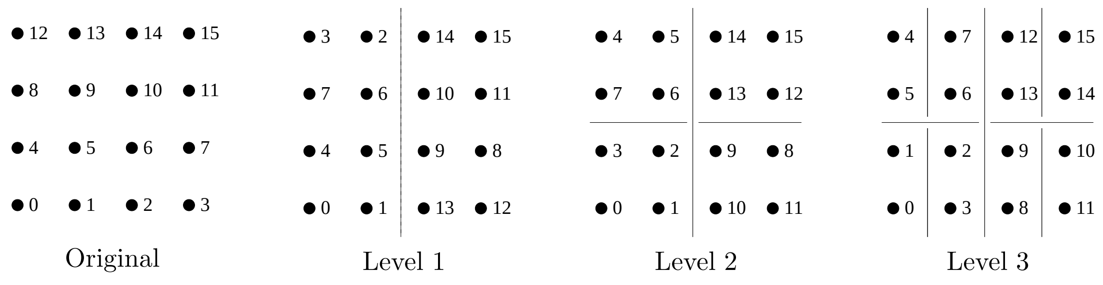
```
static pcluster
split_geometrically(pclusterfactory factory, int *index,
                    int start, int sz, int leafsize)
{
  /* ... some initialization ... */

  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 regular clustering strategy, 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 2.10 (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 2.11 (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 `tau`, we can iterate over all corresponding intervals by using the following simple loop:

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

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

```
rk = new_rkmatrix(k, tau->size, sigma->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 `tau` corresponds to the basis function with the number `dof2idx[tau->start]`, not to that with the number `tau->start`. Let us consider an example: if we want to implement the function `fill_fullmatrix` from Exercise 4, we have to make sure that we use the intervals `dof2idx[i]` and `dof2idx[j]` to compute M_{ij}, not the intervals `i` and `j`.

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 2.12 (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 2.13 (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*

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

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

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

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

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

Definition 2.14 (Homogeneity) *A block cluster tree $T_{\mathcal{I}\times\mathcal{J}}$ for $T_{\mathcal{I}}$ and $T_{\mathcal{J}}$ will be called* homogeneous*, if*

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

holds for all $t \times s \in T_{\mathcal{I}\times\mathcal{J}}$.

If a block cluster tree is homogeneous, we have

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

for all $t \times s \in T_{\mathcal{I}\times\mathcal{J}}$, 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

$$\min\{\operatorname{diam}(\Omega_t), \operatorname{diam}(\Omega_s)\} \leq \eta \operatorname{dist}(\Omega_t, \Omega_s), \tag{2.2}$$

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 $t \times s$ 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 2.15 (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 some admissibility condition if*

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

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

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 (cf. Subsection 2.3).

Checking the condition (2.2) 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.

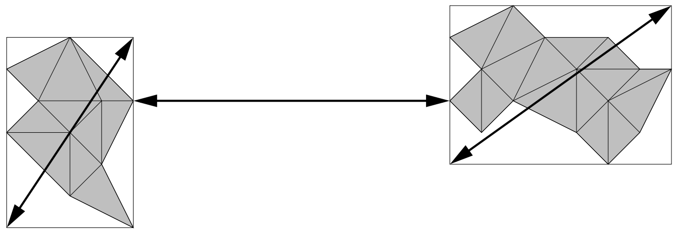
2.2.3 Bounding boxes

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 (2.2) by the larger boxes Q_t and Q_s, we get the admissibility condition

$$\min\{\operatorname{diam}(Q_t), \operatorname{diam}(Q_s)\} \leq \eta \operatorname{dist}(Q_t, Q_s). \tag{2.3}$$

This condition obviously implies (2.2).



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.

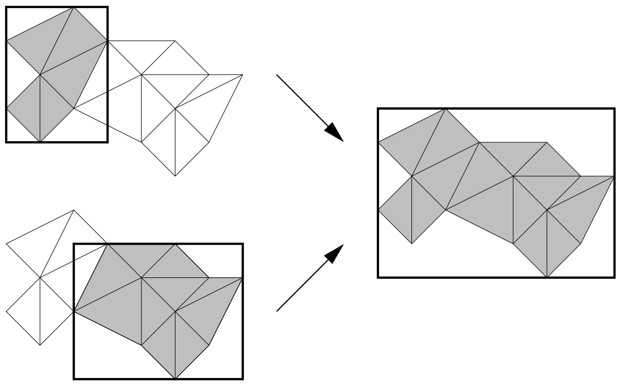
2.2.4 Implementation

Implementation 2.16 (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 2.17 (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* `nn` *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 2.18 (Curve in 2D, support) *In the Example 2.10, 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 2.19 (FE grid in 2D, support) *Creating bounding boxes in the setting of Example 2.11 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 2.20 (`blockcluster`) *The* `blockcluster` *structure is defined as follows:*

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

struct _blockcluster {
  pccluster row;
  pccluster col;
  unsigned 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 field* `type` *contains information about the admissibility criteria this block satisfies:* `0` *means that it is not admissible,* `1` *corresponds to weak admissibility,* `3` *means that (2.3) holds (for some parameter* η*), and* `7` *means that the strong admissibility condition (7.14) holds (again for some parameter* η*). Other variants are also possible, please check* `blockcluster.h` *for details.*

The following simple algorithm constructs a `blockcluster` structure from `cluster` structures and the standard admissibility condition (2.3):

```
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);

  if(diam_row < eta*dist || diam_col < eta*dist)
    bc = new_blockcluster(row, col, 0, 0,
                          HLIB_BLOCK_MINADM | HLIB_BLOCK_WEAKADM);
  else if(row->sons > 0 && col->sons > 0) {
    bc = new_blockcluster(row, col, row->sons, col->sons, 0);
    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, 0);

  return bc;
}
```

Implementation 2.21 (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.*

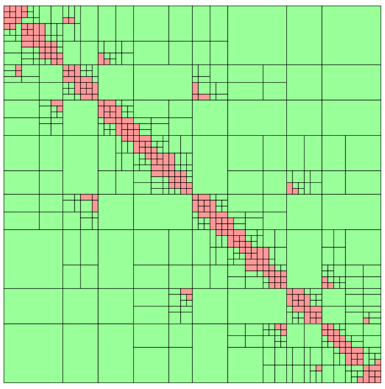
2.3 Construction of an admissible `supermatrix` structure

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 & HLIB_BLOCK_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 2.10 and a block cluster tree constructed for the standard admissibility condition (2.3), we get a `supermatrix` structure that corresponds to the following matrix partition:



2.4 Exercises

2.4.1 Theory

Exercise 7 *Let T 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 $t \in T$.*

2.4.2 Practice

Exercise 8 *The routine* `build_blockcluster` *stops splitting matrix blocks as soon as one of the corresponding clusters is a leaf.*

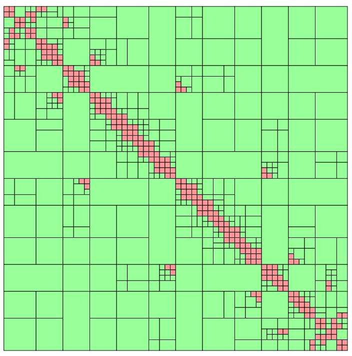
Write a routine `build_blockcluster_inhom` *that splits the blocks until* both *clusters are leaves. If a pair (τ, σ) is not admissible, we distinguish four cases:*

- *If $\operatorname{sons}(\tau) \neq \emptyset$ and $\operatorname{sons}(\sigma) \neq \emptyset$, examine all pairs (τ', σ') with $\tau' \in \operatorname{sons}(\tau)$ and $\sigma' \in \operatorname{sons}(\sigma)$.*

- *If* $\mathrm{sons}(\tau) = \emptyset$ *and* $\mathrm{sons}(\sigma) \neq \emptyset$, *examine all pairs* (τ, σ') *with* $\sigma' \in \mathrm{sons}(\sigma)$.
- *If* $\mathrm{sons}(\tau) \neq \emptyset$ *and* $\mathrm{sons}(\sigma) = \emptyset$, *examine all pairs* (τ', σ) *with* $\tau' \in \mathrm{sons}(\tau)$.
- *If* $\mathrm{sons}(\tau) = \emptyset$ *and* $\mathrm{sons}(\sigma) = \emptyset$, *create a full matrix.*

Differently from `build_blockcluster`, *this new routine may create an inhomogeneous block cluster tree.*

Running the program `example_cluster.c` *for* 64 *basis functions should give you a file* `matrix.ps` *containing the following block partition:*



Chapter 3

Integral Equations

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

The basic approach is to replace the kernel by a degenerate expansion [26], this leads to the *panel-clustering method*. One of these expansions is the multipole expansion [30, 19] for the Laplace kernel.

We use a relatively general approach based on polynomial interpolation that has been described in [3]. Its implementation is quite simple and it can be applied to all *asymptotically smooth* (cf. (3.14)) kernel functions.

3.1 Galerkin discretization

We consider a general integral equation

$$\mathcal{G}u + \lambda\langle u, \cdot\rangle = f \tag{3.1}$$

in a Hilbert space H, where $\mathcal{G}$ is an integral operator mapping 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

$$a(u, v) + \lambda m(u, v) = f(v) \tag{3.2}$$

for all $v \in H$, where

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

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

$$a(u, v) = \int_\Omega v(x) \int_\Omega g(x, y)u(y)\,\mathrm{d}y\,\mathrm{d}x \tag{3.3}$$

for a kernel function $g(\cdot, \cdot)$ and domains or manifolds Ω.

The equation (3.2) is discretized 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

$$a(u_n, v_n) + \lambda m(u_n, v_n) = f(v_n)$$

holds for all $v_n \in H_n$. For any basis $(\varphi_i)_{i\in\mathcal{I}}$ of H_n, this is equivalent to

$$a(u_n, \varphi_i) + \lambda m(u_n, \varphi_i) = f(\varphi_i)$$

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

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

so the coefficients satisfy the equation

$$\sum_{j\in\mathcal{I}} x_j a(\varphi_j, \varphi_i) + \lambda \sum_{j\in\mathcal{I}} m(\varphi_j, \varphi_i) = f(\varphi_i)$$

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

$$G_{ij} := a(\varphi_j, \varphi_i) = \int_\Omega \varphi_i(x) \int_\Omega g(x,y)\varphi_j(y)\,\mathrm{d}y\,\mathrm{d}x, \tag{3.4}$$

$$M_{ij} := m(\varphi_j, \varphi_i) = \int_\Omega \varphi_i(x)\varphi_j(x)\,\mathrm{d}x \quad \text{and} \tag{3.5}$$

$$b_i := f(\varphi_i). \tag{3.6}$$

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 $k(\cdot,\cdot)$ by local degenerate approximations, and this leads to a hierarchical matrix.

3.2 Interpolation

3.2.1 Degenerate approximation

In Section 1.2, we have used the approximation

$$\tilde{g}(x,y) = \sum_{\nu=0}^{k-1} \frac{1}{\nu!} \partial_x^\nu g(x_0, y)(x - x_0)^\nu \tag{3.7}$$

of the kernel function $g(\cdot,\cdot)$. This approach works only if the derivatives of the kernel function can be evaluated efficiently. In order to avoid this restriction, we use interpolation instead of the Taylor expansion to construct an approximation: let $(x_\nu)_{\nu\in K}$ be a family of interpolation points in $\mathbb{R}^d$, and let $(\mathcal{L}_\nu)_{\nu\in K}$ be the corresponding Lagrange functions satisfying

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

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

$$\tilde{g}(x,y) := \sum_{\nu\in K} g(x_\nu, y)\mathcal{L}_\nu(x). \tag{3.8}$$

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

Using this approximation, the matrix G from (3.4) is replaced by a matrix $\tilde{G}$ defined by

$$\tilde{G}_{ij} := \int_\Omega \varphi_i(x) \int_\Omega \tilde{g}(x,y)\varphi_j(y)\,\mathrm{d}y\,\mathrm{d}x = \sum_{\nu\in K} \int_\Omega \varphi_i(x)\mathcal{L}_\nu(x)\,\mathrm{d}x \int_\Omega \varphi_j(y) g(x_\nu, y)\,\mathrm{d}y = (AB^\top)_{ij}, \tag{3.9}$$

where we set

$$A_{i\nu} := \int_\Omega \varphi_i(x)\mathcal{L}_\nu(x)\,\mathrm{d}x \quad \text{and} \quad B_{j\nu} := \int_\Omega \varphi_j(y) g(x_\nu, y)\,\mathrm{d}y. \tag{3.10}$$

Obviously, the rank of $\tilde{G} = AB^\top$ is bounded by $\#K$, so we have indeed found an alternative method for computing low-rank approximations.

The computation of the entries of A and B is simple: in the matrix A, only piecewise polynomials have to be integrated. This can be done by any exact quadrature rule. For the computation of the matrix B, we can use quadrature rules or try to find symbolic expressions for the integrals (cf. Exercise 9).

3.2.2 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 functions 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

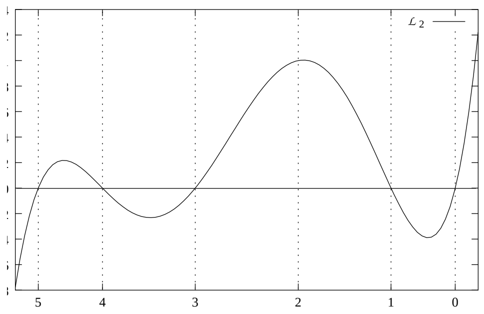
$$(x_\nu)_{\nu=0}^m = \left(\cos\left(\frac{2\nu+1}{2m+2}\pi\right)\right)_{\nu=0}^m$$

are a good choice. The Lagrange polynomials have the form

$$\mathcal{L}_\nu(x) = \prod_{\mu=0,\mu\neq\nu}^m \frac{x - x_\mu}{x_\nu - x_\mu},$$

and the corresponding interpolation operator is given by

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



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

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

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

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

It can be written in the form

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

so it is straightforward to define the transformed interpolation points

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

and corresponding Lagrange functions

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

We observe that

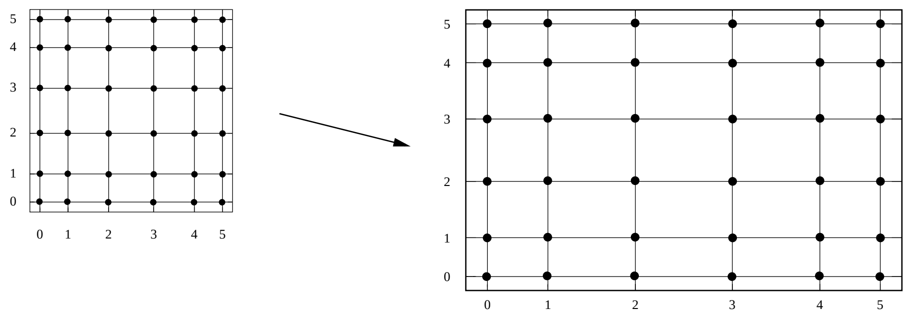
$$\mathcal{L}_\nu^{[a,b]}(x_\mu^{[a,b]}) = \mathcal{L}_\nu \circ \Phi_{[a,b]}^{-1}(\Phi_{[a,b]}(x_\mu)) = \mathcal{L}_\nu(x_\mu) = \delta_{\nu\mu}$$

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

$$\mathcal{L}_\nu^{[a,b]}(x) = \prod_{\mu=0,\mu\neq\nu}^{m} \frac{x - x_\mu^{[a,b]}}{x_\nu^{[a,b]} - x_\mu^{[a,b]}}.$$

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

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

of multi-indices and the corresponding interpolation points and Lagrange polynomials

$$x_\nu^t := (x_{\nu_1}^{[a_1,b_1]}, \ldots, x_{\nu_d}^{[a_d,b_d]}), \quad \mathcal{L}_\nu^t := \mathcal{L}_{\nu_1}^{[a_1,b_1]} \otimes \cdots \otimes \mathcal{L}_{\nu_d}^{[a_d,b_d]}, \tag{3.11}$$

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

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

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

$$\mathcal{L}_\nu^t(x) = \left(\mathcal{L}_{\nu_1}^{[a_1,b_1]} \otimes \cdots \otimes \mathcal{L}_{\nu_d}^{[a_d,b_d]}\right)(x) = \prod_{i=1}^{d} \mathcal{L}_{\nu_i}^{[a_i,b_i]}(x_i) = \prod_{i=1}^{d} \prod_{\mu=0,\mu\neq\nu_i}^{m} \frac{x_i - x_\mu^{[a_i,b_i]}}{x_{\nu_i}^{[a_i,b_i]} - x_\mu^{[a_i,b_i]}}. \tag{3.12}$$

3.2.3 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 (3.9), so we have to compute the matrices $A^{t,s}$ and $B^{t,s}$:

$$A^{t,s}_{i\nu} = \int_\Omega \varphi_i(x)\mathcal{L}^t_\nu(x)\,\mathrm{d}x, \quad B^{t,s}_{j\nu} = \int_\Omega \varphi_j(y) g(x^t_\nu, y)\,\mathrm{d}y,$$

where x^t_ν and $\mathcal{L}^t_\nu$ are the transformed interpolation points and Lagrange polynomials defined in (3.11).

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:

$$A^{t,s}_{i\nu} = \int_\Omega \varphi_i(x) g(x, x^s_\nu)\,\mathrm{d}x, \quad B^{t,s}_{j\nu} = \int_\Omega \varphi_j(y)\mathcal{L}^s_\nu(y)\,\mathrm{d}y.$$

In both cases, we need the transformed interpolation points. Given an array `xp` of dimension `p` $= m+1$ containing the points $(x_i)_{i=0}^m$ 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 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[i][j] = 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 x^t_ν is then given by `l[nu[j]][j]` for `nu[j]=` ν_j. Computing integrals of the Lagrange polynomials can be done 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 (3.12), this can be done by 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[i][j]) / (l[nu[j]][j] - l[i][j]);
```

3.2.4 Interpolation error bound

One-dimensional estimates

The Chebyshev interpolation operators $\mathfrak{I}_m$ satisfy the *stability estimate*

$$\|\mathfrak{I}_m f\|_{\infty,[-1,1]} \leq \Lambda_m \|f\|_{\infty,[-1,1]}$$

for the constant

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

For a function $f \in C^{m+1}[-1,1]$, the *approximation error bound*

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

holds. For the transformed interpolation operator, this implies

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

Multi-dimensional estimates

Let $t \in T_{\mathcal{I}}$ be a cluster with the bounding box $Q^t := [a_1, b_1] \times \cdots \times [a_d, b_d]$. For $i \in \{0, \ldots, d\}$, we introduce the operators

$$P_i := \mathfrak{I}_m^{[a_1,b_1]} \otimes \cdots \otimes \mathfrak{I}_m^{[a_i,b_i]} \otimes I \otimes \cdots \otimes I.$$

We have $P_0 = I$ and $P_d = \mathfrak{I}_m^t$ and

$$\begin{aligned}\|f - \mathfrak{I}_m^t f\|_{\infty,Q^t} &\le \sum_{i=1}^d \|P_{i-1}f - P_i f\|_{\infty,Q^t} = \sum_{i=1}^d \|P_{i-1}(I - I\otimes\cdots\otimes I\otimes\mathfrak{I}_m^{[a_i,b_i]}\otimes I\otimes\cdots\otimes I)f\|_{\infty,Q^t}\\ &\le (m+1)^{i-1}\sum_{i=1}^d \|(I - I\otimes\cdots\otimes I\otimes\mathfrak{I}_m^{[a_i,b_i]}\otimes I\otimes\cdots\otimes I)f\|_{\infty,Q^t}\\ &\le \frac{(m+1)^{d-1}}{2^{2m+1}(m+1)!}\sum_{i=1}^d (b_i - a_i)^{m+1}\|\partial_i^{m+1} f\|_{\infty,Q^t}\\ &\le \frac{d(m+1)^{d-1}}{2^{2m+1}(m+1)!}\operatorname{diam}(Q^t)^{m+1}\max\{\|\partial_i^{m+1} f\|_{\infty,Q^t} \;:\; i\in\{1,\ldots,d\}\}. \qquad (3.13)\end{aligned}$$

Application to the kernel function

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

$$|\partial_x^\alpha \partial_y^\beta g(x,y)| \le C(\alpha+\beta)!c_0^{|\alpha|+|\beta|}\|x-y\|^{-|\alpha|-|\beta|-\sigma} \qquad (3.14)$$

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

$$\begin{aligned} g_x &: Q^t \to C^\infty(Q^s), \quad x \mapsto g(x,\cdot),\\ g_y &: Q^s \to C^\infty(Q^t), \quad y \mapsto g(\cdot,y),\end{aligned}$$

satisfy the estimates

$$\begin{aligned}\|\partial^\alpha g_x(x)\|_{\infty,Q^t} &\le C\alpha! c_0^{|\alpha|}\operatorname{dist}(Q^t,Q^s)^{-|\alpha|-\sigma},\\ \|\partial^\beta g_y(y)\|_{\infty,Q^s} &\le C\beta! c_0^{|\beta|}\operatorname{dist}(Q^t,Q^s)^{-|\beta|-\sigma}\end{aligned}$$

for $x \in Q^t$ and $y \in Q^s$, so we can apply the interpolation estimate (3.13) in order to get

$$\begin{aligned}\|g_x - \mathfrak{I}_m^t g_x\|_{\infty,Q^t} &\le \frac{Cd(m+1)^{d-1}}{2\operatorname{dist}(Q^t,Q^s)^\sigma}\left(\frac{c_0\operatorname{diam}(Q^t)}{4\operatorname{dist}(Q^t,Q^s)}\right)^{m+1}\\ \|g_y - \mathfrak{I}_m^s g_y\|_{\infty,Q^s} &\le \frac{Cd(m+1)^{d-1}}{2\operatorname{dist}(Q^t,Q^s)^\sigma}\left(\frac{c_0\operatorname{diam}(Q^s)}{4\operatorname{dist}(Q^t,Q^s)}\right)^{m+1}.\end{aligned}$$

If $\operatorname{diam}(Q^t) \le \operatorname{diam}(Q^s)$, we have

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

In a similar fashion, we can treat the case $\operatorname{diam}(Q^s) \leq \operatorname{diam}(Q^t)$ and conclude

$$|g(x,y) - \tilde{g}(x,y)| \leq C'(m) \left(\frac{c_0\eta}{4}\right)^{m+1} \operatorname{dist}(Q^t, Q^s)^{-\sigma} \tag{3.15}$$

for the polynomial $C'(m) := Cd(m+1)^d/2$. 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 3.1 (Improved error bound) *In [3], 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.*

3.3 Example: Boundary Element Method in 2D

In Example 2.10, 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.

3.3.1 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 3.2 (Curve integral) *Let $\epsilon \in \mathbb{R}_{>0}$. 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*

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

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

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

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

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

■

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 $i \in \{1,\dots,m\}$, 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

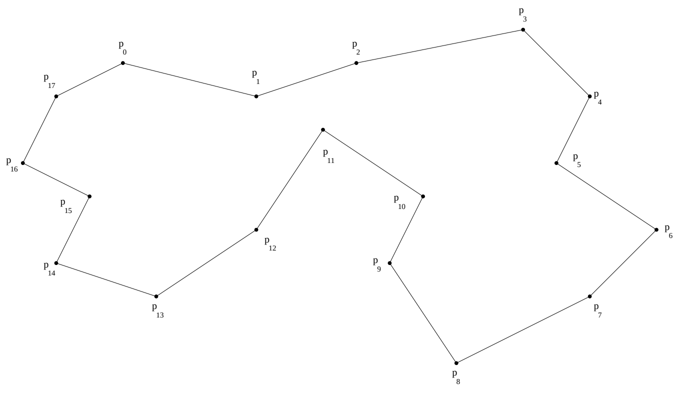
$$\int_\Gamma u(x)dx := \sum_{i=1}^m \int_0^1 u(\gamma_i(y))\|\gamma_i'(y)\|\,\mathrm{d}y.$$

3.3.2 Single layer potential

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

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

for $i \in \{1,\dots,n\}$. As long as $p_i \neq p_j$ holds for all $i,j \in \{0,\dots,n-1\}$ 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 discretize $a_{\mathrm{slp}}(\cdot,\cdot)$ by piecewise constant functions $(\varphi_i)_{i=1}^n$ defined through

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

for $i,j \in \mathcal{I} := \{1,\dots,n\}$. 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 Subsection 3.2.1 and can construct a hierarchical matrix by replacing the logarithmic kernel $g(x,y) := \log(\|x-y\|)$ by degenerate approximations.

3.3.3 Implementation

We have already considered the construction of suitable cluster trees and admissible partitions in Examples 2.10 and 2.18, 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 initializes 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 $\operatorname{diam}(Q^t) \le \operatorname{diam}(Q^s)$, 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

$$A_{i\nu}^{t,s} = \int_\Gamma \varphi_i(x)\mathcal{L}_\nu^t(x)\,\mathrm{d}x = \|p_i-p_{i-1}\|\int_0^1 \mathcal{L}_\nu^t(\gamma_i(x))\,\mathrm{d}x, \tag{3.16}$$

$$B_{j\nu}^{t,s} = -\frac{1}{2\pi}\int_\Gamma \varphi_j(y)\log(\|x_\nu^t-y\|)\,\mathrm{d}y = -\frac{1}{2\pi}\|p_j-p_{j-1}\|\int_0^1 \log(\|x_\nu^t-\gamma_j(y)\|)\,\mathrm{d}y. \tag{3.17}$$

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 3.2.3.

Filling the matrix $B^{s,t}$ 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 9). 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 initialized 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 initialize 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`.

3.4 Exercises

3.4.1 Practice

Exercise 9 *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 10 (BEM) *The structure* `curvebemfactory` *contains the data necessary for the discretization 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 initialize the matrices `X` *as follows: for the first function, the matrix* `X` *is filled with the entries of the matrix* $A^{t,s}$ *from* (3.16). *For the second function, it is filled with the entries of the matrix* $B^{t,s}$.

Chapter 4

Elliptic Partial Differential Equations

In the previous two sections we have introduced the hierarchical matrix format that is well suited for the fast assembly and evaluation of stiffness matrices that stem from non-local integral operators. Since the stiffness matrix of those operators is in general dense, one has to apply some compression method in order to avoid the $\mathcal{O}(n^2)$ complexity.

The story is different for the discretisation and solution of partial differential equations. Such an operator is typically 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 Section 4.3 we prove that the inverse allows for a data-sparse $\mathcal{H}$-matrix approximation. In this chapter we consider the one-dimensional case (cf. [23]) where algebraic arguments allow us to give a simple proof.

4.1 One-Dimensional Model Problem

We consider the differential equation

$$-\mathcal{U}''(x) = \mathcal{F}(x), \qquad x \in [0,1], \tag{4.1}$$

with Dirichlet boundary conditions

$$\mathcal{U}(0) = 0, \qquad\qquad \mathcal{U}(1) = 0 \tag{4.2}$$

for a suitable right-hand side $\mathcal{F} : [0,1] \to \mathbb{R}$ and seek the solution $\mathcal{U} : [0,1] \to \mathbb{R}$.

4.1.1 Discretization

A standard discretisation scheme is Galerkin's method (see [21]) where we solve equation (4.1) projected onto the (n-dimensional) space $V_n := \operatorname{span}\{\varphi_0, \dots, \varphi_{n-1}\}$,

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

Partial integration (and the zero boundary condition) yields

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

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

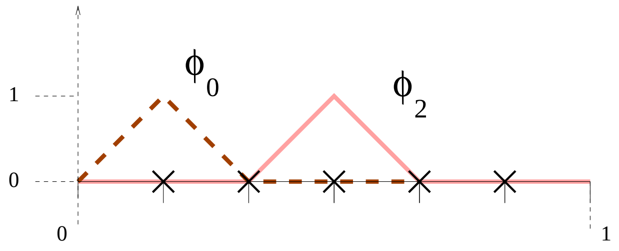
$$Gu = f, \quad G_{ij} := \int_0^1 \varphi_i'(x)\varphi_j'(x)\mathrm{d}x, \quad f_i := \int_0^1 \mathcal{F}(x)\varphi_i(x)\mathrm{d}x.$$

In the introductory example from Section 1.1 we have chosen piecewise constant basis functions but here, the partial integration forbids the use of such functions.

Instead, we choose piecewise affine and continuous basis functions

$$\varphi_i(x) = \begin{cases} (n+1)(x - \frac{i-1}{n+1}) & \text{if } \frac{i-1}{n+1} \le x \le \frac{i}{n+1} \\ (n+1)(\frac{i+1}{n+1} - x) & \text{if } \frac{i}{n+1} \le x \le \frac{i+1}{n+1} \\ 0 & \text{otherwise} \end{cases} \tag{4.3}$$

in the interior of a uniform grid of $[0,1]$ with step size $h = 1/(n+1)$:



The stiffness matrix G is sparse in the sense that in each row at most three entries are nonzero. Basic calculus shows

$$G = (n+1)\begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{bmatrix}.$$

Contrary to the integral equation from Section 1.1, the assembly of the entries for the stiffness matrix G is not difficult. Also, the matrix is already sparse and needs no compression. The matrix vector multiplication can be performed in optimal complexity $\mathcal{O}(n)$. However, the inverse G^{-1} to G is not sparse. Its computation as well as storing it is complicated.

4.1.2 The Stiffness Matrix in $\mathcal{H}$-Matrix Format

For the ease of presentation we assume that n is a power of 2:

$$n = 2^p.$$

The stiffness matrix G can easily be written as a 2×2 block matrix

$$G = (n+1)\left[\begin{array}{c|c} G' & -1 \\ \hline -1 & G' \end{array}\right],$$

where G' denotes the stiffness matrix for the model problem (4.1) with $n/2$ degrees of freedom but scaled by $\frac{2n}{n+1}$. The two off-diagonal blocks (with the entry "-1") are of rank 1. The two diagonal blocks can be split up in the same way. Successively each diagonal block can be split up until we reach the cardinality 1 (see Figure 4.1). The result is a hierarchical blocking of the matrix or, in other words, an $\mathcal{H}$-matrix. The cluster tree $T_{\mathcal{I}}$, $\mathcal{I} := \{0, \dots, n-1\}$, is analogous to the one in Section 1.4. The block cluster tree $T_{\mathcal{I}\times\mathcal{I}}$ is defined by

$$\begin{aligned} \operatorname{root}(T_{\mathcal{I}\times\mathcal{I}}) &:= \mathcal{I} \times \mathcal{I}, \\ \operatorname{sons}(r \times s) &:= \begin{cases} \{r' \times s' \mid r' \in \operatorname{sons}(r), s' \in \operatorname{sons}(s)\} & \text{if } r = s, \\ \emptyset & \text{otherwise.} \end{cases} \end{aligned}$$

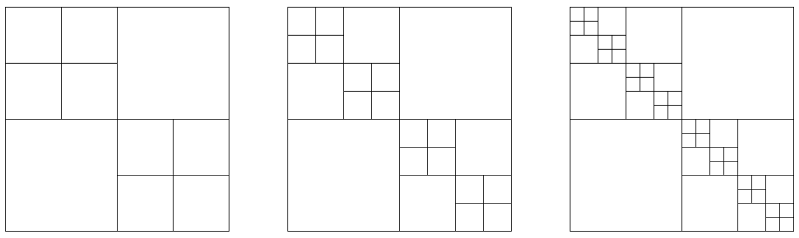


Figure 4.1: The block structure of the stiffness matrix for $p = 2, 3, 4$.

The matrix G is an $\mathcal{H}$-matrix based on the tree $T_{\mathcal{I}\times\mathcal{I}}$ with blockwise rank $k = 1$:

$$G \in \mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, 1).$$

4.1.3 The Inverse to the Stiffness Matrix in $\mathcal{H}$-matrix Format

Inversion of a 2×2 Block Matrix

Let the matrix M be given in 2×2 block form

$$M = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix}$$

with M, M_{11} regular. We perform a series of regular transformations to the equation $MM^{-1} = I$:

$$\begin{aligned}
\begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix} M^{-1} &= \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix} \\
\Rightarrow \begin{bmatrix} I & M_{11}^{-1}M_{12} \\ M_{21} & M_{22} \end{bmatrix} M^{-1} &= \begin{bmatrix} M_{11}^{-1} & 0 \\ 0 & I \end{bmatrix} \\
\Rightarrow \begin{bmatrix} I & M_{11}^{-1}M_{12} \\ 0 & M_{22} - M_{21}M_{11}^{-1}M_{12} \end{bmatrix} M^{-1} &= \begin{bmatrix} M_{11}^{-1} & 0 \\ -M_{21}M_{11}^{-1} & I \end{bmatrix} \\
\Rightarrow \begin{bmatrix} I & M_{11}^{-1}M_{12} \\ 0 & I \end{bmatrix} M^{-1} &= \begin{bmatrix} M_{11}^{-1} & 0 \\ -(M_{22} - M_{21}M_{11}^{-1}M_{12})^{-1}M_{21}M_{11}^{-1} & (M_{22} - M_{21}M_{11}^{-1}M_{12})^{-1} \end{bmatrix} \\
\Rightarrow \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix} M^{-1} &= \begin{bmatrix} M_{11}^{-1} + M_{11}^{-1}M_{12}S^{-1}M_{21}M_{11}^{-1} & -M_{11}^{-1}M_{12}S^{-1} \\ -S^{-1}M_{21}M_{11}^{-1} & S^{-1} \end{bmatrix}
\end{aligned}$$

with $S := M_{22} - M_{21}M_{11}^{-1}M_{12}$. The formula in the last row,

$$M^{-1} = \begin{bmatrix} M_{11}^{-1} + M_{11}^{-1}M_{12}S^{-1}M_{21}M_{11}^{-1} & -M_{11}^{-1}M_{12}S^{-1} \\ -S^{-1}M_{21}M_{11}^{-1} & S^{-1} \end{bmatrix}, \tag{4.4}$$

is sometimes called *Frobenius formula* but basically it is obtained by block Gauss elimination and can be generalised to arbitrary $m \times m$ block matrices. We will only need the formula for the 2×2 case.

Inversion of a Tridiagonal Matrix

The stiffness matrix G from Section 4.1 can be regarded as a (hierarchical) 2×2 block matrix. Additionally, it is a tridiagonal matrix which can be exploited in the inversion formula (4.4).

Lemma 4.1 *Let $n = 2^p$ and $M \in \mathbb{R}^{n\times n}$ be a tridiagonal matrix. We assume that all principal matrices of M are regular (this is needed for the Gaussian elimination). Then*

$$M^{-1} \in \mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, 1)$$

for the block cluster tree $T_{\mathcal{I}\times\mathcal{I}}$ from Section 4.1.2. Recursive application of (4.4) yields the exact inverse M^{-1}.

Proof: We prove the statement by induction. For $p = 0$ and $n = 2^p = 1$ the statement holds. Now let

$$M = \begin{bmatrix} M^{(11)} & M^{(12)} \\ M^{(21)} & M^{(22)} \end{bmatrix}.$$

The off-diagonal blocks of M^{-1} are

$$(M^{-1})^{(12)} \overset{(4.4)}{=} -(M^{(11)})^{-1}M^{(12)}S^{-1}$$

and since $\operatorname{rank}(M^{(12)}) = 1$ we get $\operatorname{rank}((M^{-1})^{(12)}) \leq 1$. The same holds for the other off-diagonal block: $\operatorname{rank}((M^{-1})_{21}) \leq 1$. The block $S = M^{(22)} - M^{(21)}(M^{(11)})^{-1}M^{(12)}$ is again tridiagonal:

$$(M^{(21)}(M^{(11)})^{-1}M^{(12)})_{ij} = \begin{cases} ((M^{(11)})^{-1})_{n-1,n-1} & \text{if } (i,j) = (n/2, n/2) \\ 0 & \text{else.} \end{cases}$$

Hence, $M^{(22)}$ and $M^{(21)}(M^{(11)})^{-1}M^{(12)}$ are both tridiagonal. By induction $(M^{-1})^{(22)} = S^{-1} \in \mathcal{H}(T_{\mathcal{I}'\times\mathcal{I}'}, 1)$ where $\mathcal{I}' := \{n/2, \ldots, n-1\}$ and $T_{\mathcal{I}'\times\mathcal{I}'}$ is the subtree of $T_{\mathcal{I}\times\mathcal{I}}$ with root $\mathcal{I}' \times \mathcal{I}'$. By analogy we also get $(M^{-1})^{(11)} \in \mathcal{H}(T_{\mathcal{I}''\times\mathcal{I}''})$ for the subtree $T_{\mathcal{I}''\times\mathcal{I}''}$ to the index set $\mathcal{I}'' := \{0, \ldots, n/2-1\}$. ■

The structure of the block cluster tree $T_{\mathcal{I}\times\mathcal{I}}$ is adapted to the tridiagonal stiffness matrix G. In higher dimensions ($d = 2, 3$) the structure has to be analogous to the block cluster tree in Section 2.1. The statement "$G^{-1} \in \mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, k)$" will no longer be true unless $k = \mathcal{O}(n)$. This means that the algebraic approach used in the previous lemma cannot be used in higher spatial dimensions.

4.2 Multi-dimensional Model Problem

Example 4.2 (Poisson's Equation) *We consider the differential equation*

$$-\Delta\mathcal{U}(x) := -\sum_{i=1}^{d} \partial_i^2 \mathcal{U}(x) = \mathcal{F}(x), \qquad x \in \Omega \subset \mathbb{R}^d, \tag{4.5}$$

with Dirichlet boundary conditions

$$\mathcal{U}(x) = 0 \qquad\qquad \text{in } x \in \Gamma := \partial\Omega \tag{4.6}$$

for a right-hand side $\mathcal{F} \in L^2(\Omega)$ and seek the solution $\mathcal{U} \in H_0^1(\Omega)$. Let $V_n := \langle \varphi_0, \ldots, \varphi_{n-1}\rangle$ be an n-dimensional subspace of $H_0^1(\Omega)$. The coefficent vector u of the Galerkin solution $\mathcal{U}_n(x) := \sum_{i=1}^{n} u_i\varphi_i(x)$ is the solution to the linear system

$$Gu = f, \qquad G_{ij} := \int_\Omega \langle \nabla\varphi_i(x), \nabla\varphi_j(x)\rangle \,\mathrm{d}x, \quad f_i := \int_\Omega \varphi_i(x)\mathcal{F}(x)\,\mathrm{d}x. \tag{4.7}$$

The matrix G is sparse and the entries can easily be computed for typical basis functions φ_i, e.g., piecewise affine basis functions on a triangulation of Ω.

For domains Ω, where the first Green's formula holds, we can represent the solution $\mathcal{U}$ by the singularity function (see [21])

$$s : \Omega \times \Omega \to \mathbb{R}, \qquad s(x,y) = \begin{cases} -\frac{1}{2\pi}\log\|x-y\| & \text{if } d = 2 \\ \frac{1}{4\pi}\|x-y\|^{-1} & \text{if } d = 3, \end{cases}$$

in the integral form

$$\mathcal{U}(x) = \int_\Omega s(x,y)\mathcal{F}(y)\mathrm{d}y - \int_\Gamma s(x,y)\partial_n\mathcal{U}(y)\mathrm{d}\Gamma_y.$$

If the domain Ω possesses a Green's function $g(x,y) = s(x,y) + \Phi(x,y)$, i.e., g is zero on the boundary Γ and Φ is harmonic in Ω and twice differentiable in $\bar{\Omega}$, then the representation of $\mathcal{U}$ simplifies to

$$\mathcal{U}(x) = \int_\Omega g(x,y)\mathcal{F}(y)\mathrm{d}y. \tag{4.8}$$

The essential behaviour of g is described by the singularity function s which is of the same type as the kernel functions considered in Chapter 3. Therefore, it is reasonable to use the same structure also for the inverse G^{-1} to the FEM stiffness matrix G.

The construction of the cluster tree $T_\mathcal{I}$, $\mathcal{I} := \{0,\dots,n-1\}$, is done as in Section 2.1 by geometrically balanced binary space partitioning until the minimal leafsize $n_{\min}$ is reached. The admissibility condition (2.3) is used to define the block cluster tree $T_{\mathcal{I}\times\mathcal{I}}$ (split only inadmissible blocks).

Lemma 4.3 ($\mathcal{H}$-matrix representation of the stiffness matrix) *The stiffness matrix G fulfils $G \in \mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, 0)$.*

Proof: Let $t \times s \in T_{\mathcal{I}\times\mathcal{I}}$ be an admissible block. Then

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

for some parameter $\eta > 0$. Since $\operatorname{diam}(\Omega_t) > 0$ (a continuous basis function cannot have a trivial support) we can conclude that $\operatorname{dist}(\Omega_t, \Omega_s) > 0$ holds, which means that the supports of the basis functions $i \in \hat{t}$ are disjoint from the supports of the basis functions $j \in \hat{s}$. Due to (4.7) we get $G_{ij} = 0$, thus $\operatorname{rank}(G|_{\hat{t}\times\hat{s}}) = 0$. ■

The previous lemma is not a surprise because the stiffness matrix G is sparse. The inverse G^{-1} on the other hand is a dense matrix and we seek an **approximation** $\widetilde{G^{-1}}$ to G^{-1} in the set $\mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, k)$ with preferably small rank k and good approximation quality.

By using Galerkin's method we have already introduced a discretisation error

$$\epsilon_n(\mathcal{U}) := \|\mathcal{U} - P_n\mathcal{U}\|_{L^2(\Omega)}$$

where P_n is the Ritz projection onto V_n. The finite element convergence is described by

$$\epsilon_n(\mathcal{U}) \leq \varepsilon_n \|\mathcal{F}\|_{L^2(\Omega)}.$$

The approximation error $\|G^{-1} - \widetilde{G^{-1}}\|_2$ should now be of the same size as the discretisation error:

Theorem 4.4 (Existence of an $\mathcal{H}$-matrix approximant) *Let $p := \operatorname{depth}(T_{\mathcal{I}\times\mathcal{I}})$. Then there exists a matrix $\widetilde{G^{-1}} \in \mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, k)$ that fulfils*

$$\|G^{-1} - \widetilde{G^{-1}}\|_2 = \mathcal{O}(\varepsilon_n)$$

for a blockwise rank

$$k = \mathcal{O}\left(p^2 \log(p/\varepsilon_n)\right).$$

The proof of the theorem does *not* yield a method to construct the approximant $\widetilde{G^{-1}}$ — it involves the unknown Green's function g from (4.8). Therefore the question remains how one can compute the approximant $\widetilde{G^{-1}}$. This is not done by an analytical approach but by algebraic manipulations explained in the next chapter.

4.3 Approximation of the inverse of the stiffness matrix

We will now prove that the inverse to the finite element stiffness matrix of an elliptic operator on a quasi-uniform grid can be approximated (up to the size of the discretisation error ε) by an $\mathcal{H}$-matrix with blockwise rank $k = \mathcal{O}(\log(1/\varepsilon)^{d+3})$. This result is contained in [1]. For integral operators we have seen in Chapter 3 that the dense stiffness matrix B allows for an efficient $\mathcal{H}$-matrix approximation $B_{\mathcal{H}}$. Here, the situation is similar, because the FEM stiffness matrix A can be written (up to the discretisation error) as

$$A^{-1} \approx M^{-1} B M^{-1},$$

where M is the FEM mass matrix and B the stiffness matrix of an integral operator whose kernel is the Green function of the underlying differential operator. Unfortunately, the Green function does not possess the nice smoothness properties we exploited in Chapter 3, therefore we have to prove a similar feature that allows us to approximate the Green function in admissible parts of the domain by a degenerate kernel.

4.3.1 Notation of Finite Elements and Introduction into the Problem

In the following, we consider an elliptic boundary-value problem of second order in a domain $\Omega \subset \mathbb{R}^d$. For simplicity, we concentrate to the principal part and use Dirichlet conditions:

$$\begin{aligned} Lu = f \text{ in } \Omega, \qquad &\text{where } Lu := -\sum_{i,j=1}^{d} \partial_j(c_{ij}\partial_i u), \qquad (4.9)\\ u = 0 \text{ on } \Gamma := \partial\Omega. \end{aligned}$$

The variational formulation is

$$a(u,v) = \int f(x)v(x)\mathrm{d}x, \qquad \text{where} \qquad (4.10)$$

$$a(u,v) := \int_\Omega \sum_{i,j=1}^{d} c_{ij}(x)\partial_i u(x)\partial_j v(x)\mathrm{d}x. \qquad (4.11)$$

The precise assumptions of the boundary-value problem are as follows:

- The domain $\Omega \subset \mathbb{R}^d$ is a bounded Lipschitz domain.
- The coefficient matrix $C = C(x) = (c_{ij}(x))_{ij}$, $c_{ij} \in L^\infty(\Omega)$, is symmetric with

$$0 < \lambda_{\min} \le \lambda \le \lambda_{\max} \qquad (4.12)$$

  for all eigenvalues λ of $C(x)$ and almost all $x \in \Omega$.

Remark 4.5 *The ratio $\kappa_C = \lambda_{\max}/\lambda_{\min}$ is an upper bound on almost all spectral condition numbers* $\mathrm{cond}_{\|\cdot\|_2} C(x)$. *This does not necessarily hold the other way round.*

In the finite element discretisation, one introduces a (d-dimensional) triangulation $\mathcal{T}$, which covers (or approximates) the domain.

For all elements $t \in \mathcal{T}$ one can consider the radius $\bar{r}(t)$ of the minimal sphere containing t and the radius $\underline{r}(t)$ of the maximal sphere contained in t. $\mathcal{T}$ is called *shape regular*, if the ratio $\bar{r}(t)/\underline{r}(t)$ remains bounded for all $t \in \mathcal{T}$ and the whole family of triangulations $\mathcal{T}$. In the case of triangles (i.e., $d = 2$), the shape regularity can also described by $\alpha_i(t) \ge \underline{\alpha} > 0$ for all angles $\alpha_1(t)$, $\alpha_2(t)$, $\alpha_3(t)$ of t.

If there is a constant c such that $\mathrm{diam}(t) \le c\,\mathrm{diam}(t')$ for all $t, t' \in \mathcal{T}$, the triangulation $\mathcal{T}$ is said to be *quasi-uniform*. In particular, we define $h := \max\{\mathrm{diam}(t) : t \in \mathcal{T}\}$ and $h_{\min}$:= minimal side length over all $t \in \mathcal{T}$ and remark that $h_{\min} \le h \le \mathrm{const} \cdot h_{\min}$.

In the following we assume that piecewise affine finite element basis functions $(\varphi_i)_{i\in\mathcal{I}}$ are chosen for the FE space V_h. Furthermore, we formulate the following statements for the case $d=2$, although the generalisation to $d>2$ is obvious.

The celebrated advantage of a finite element discretisation is that the stiffness matrix A is sparse. The entries

$$A_{ij} := a\left(\varphi_j, \varphi_i\right) \qquad (i,j\in\mathcal{I})$$

vanish for all nodal points $i\neq j$ not belonging to the same triangle $t\in\mathcal{T}$. However, the inverse A^{-1} is a dense matrix and requires the efficient $\mathcal{H}$-matrix techniques.

The critical question is whether the inverse A^{-1} can be treated by the $\mathcal{H}$-matrix technique in the same way as the discretisations of the integral operators. A positive indication is given by the fact that A^{-1} can be considered as the discrete Green function and is closely connected to the integral operator

$$\mathcal{G}f := \int_\Omega g(x,y)f(y)\mathrm{d}y, \tag{4.13}$$

where $g(\cdot,\cdot)$ is the Green function defined by $Lg(\cdot,y)=\delta_y$ (Dirac's function at $y\in\Omega$) with boundary data $g(x,y)=0$ $(x\in\Gamma)$.

However, there is a possible obstacle: in the case of the integral operators arising from the boundary element method we make use of the fact that the kernel function is analytic (or asymptotically smooth), when we apply the Taylor series. This is not true in the case of $\mathcal{G}$. Even if $L=-\Delta$, re-entrant corners of Ω cause the well-known corner singularities of g. The most interesting case, however, are boundary-value problems with non-smooth coefficients c_{ij} in (4.9), e.g., piecewise constant ones, which are necessarily discontinuous. For nonsmooth coefficients also g has low regularity. Nevertheless, we shall show that A^{-1} enjoys the $\mathcal{H}$-matrix properties.

For this purpose we need some auxiliary results. Next, in §4.3.2, we investigate the mass matrix and its inverse. Then, in §4.3.3 we show the connection between the Galerkin discretisation of $\mathcal{G}$ from (4.13) and the inverse A^{-1} of the FE stiffness matrix. The central part of the proof is given in §4.3.4 and discusses the approximation error of

$$g(x,y) \approx \sum_{i=1}^{k} u_i(x)v_i(y) \qquad \text{in } D_1\times D_2 \tag{4.14}$$

depending on k (D_1, D_2 are subsets of Ω satisfying the admissibility condition). The result shows that the discretisation of $\mathcal{G}$ has the desired properties although g is not smooth.

4.3.2 Mass Matrices

Properties of Mass Matrices

The *mass matrix* is the Galerkin approximation of the identity operator:

$$M = \left(M_{ij}\right)_{i,j\in\mathcal{I}}, \qquad M_{ij} := \int_\Omega \varphi_i(x)\varphi_j(x)\mathrm{d}x.$$

Obvious properties of M are collected in

Remark 4.6 *The mass matrix M is always positive definite and sparse.*

The positive definiteness is a consequence of the linear independence of the φ_i's, while the sparsity follows from the fact that finite element basis functions have small local supports.

A proof for the following basic Lemma can be found in [21]:

Lemma 4.7 *Let $\mathcal{T}$ be shape regular and quasi-uniform. Then the mass matrix M is well-conditioned, i.e., the ratio $\mu_{\max}/\mu_{\min}$ of the extreme eigenvalues is bounded independently of the mesh size (or matrix size).*

There is a direct connection between the mass matrix and estimates of the norms in $L^2(\Omega)$ and $\mathbb{R}^n$. We use the notation J for the natural bijection

$$J : \mathbb{R}^n \to V_h, \qquad x \mapsto \sum_{i \in \mathcal{I}} x_i \varphi_i, \qquad \mathcal{I} = \{1, \ldots, n\}.$$

Using the adjoint J^* of J, we can write the mass matrix as $M = J^* J$.

For quasi-uniform and shape-regular triangulations it is known that there are constants $0 < c_{J,1} \leq c_{J,2}$ (independent of h and n) such that

$$c_{J,1} \|x\|_h \leq \|Jx\|_{L^2(\Omega)} \leq c_{J,2} \|x\|_h \qquad \text{for all } x \in \mathbb{R}^n, \tag{4.15}$$

provided that the discrete Euclidean norm $\|\cdot\|_h$ is suitably scaled. Obviously, $c_{J,2}$ is an upper bound of $\|J\|$; hence,

$$\begin{aligned} \mu_{\max} &:= \|M\|_2 = \|J^* J\|_2 = \|J\|^2 \leq c_{J,2}^2, \\ \mu_{\min} &:= 1/\|M^{-1}\|_2 \geq c_{J,1}^2, \\ \operatorname{cond}_2(M) &= \mu_{\max}/\mu_{\min} \leq \left(c_{J,2}/c_{J,1}\right)^2. \end{aligned}$$

Vice versa, scaling the Euclidean norm $\|\cdot\|_h$ appropriately, (4.15) holds with $c_{J,1} = \sqrt{\mu_{\min}}$ and $c_{J,2} = \sqrt{\mu_{\max}}$. We mention that the FE stiffness matrix can be written as $A = J^* L J$, where $L : H_0^1(\Omega) \to H^{-1}(\Omega)$ is the differential operator from (4.9).

The matrix graph of M consists of all nodal points (corner points of the triangles $t \in \mathcal{T}$), while an edge (P, P') exists in the graph if and only if $P, P' \in t$ for some $t \in \mathcal{T}$ (see Hackbusch [22, Section 6.2]). For $i, j \in \mathcal{I}$ we define δ_{ij} to be the minimal length of a path from i to j, where $\delta_{ii} := 0$ for $i = j$. If no path from i to j exists (e.g., for a reducible matrix M), we formally set $\delta_{ij} := \infty$. δ_{ij} defines a distance in the vertex set of the graph.

Lemma 4.8 *Let $\mathcal{T}$ be shape regular and quasi-uniform with $h_{\min}$ being the minimal side length of the triangles $t \in \mathcal{T}$. Let $\Omega_i = \operatorname{supp} \varphi_i$. Then $\delta_{ij} \geq 1 + d_{ij}/h_{\min}$ for all $i \neq j$, where $d_{ij} := \operatorname{dist}(\Omega_i, \Omega_j)$.*

Proof: Consider a path connecting the nodal points $P_i \in \Omega_i$ and $P_j \in \overline{\Omega}_j$ consisting of δ_{ij} edges. Since $i \neq j$, the first step of the path leads from $P_i \in \Omega_i$ to $Q_i \in \partial\Omega_i$. The remaining path has the length $\delta_{ij} - 1$. Since each edge of the graph corresponds to a side of length $\geq h_{\min}$, the geometrical length of the path is $\geq (\delta_{ij} - 1)\, h_{\min}$ and must be larger than $\operatorname{dist}(Q_i, P_j)$, i.e., $\delta_{ij} - 1 \geq \operatorname{dist}(Q_i, P_j)/h_{\min}$. Note that $d_{ij} := \operatorname{dist}(\Omega_i, \Omega_j) \leq \operatorname{dist}(Q_i, P_j)$. Hence, the desired estimate $\delta_{ij} \geq 1 + d_{ij}/h_{\min}$ follows. ■

Componentwise Estimates of Inverse Matrices

Let $\sigma(M)$ denote the spectrum of M, while δ_{ij} is the graph distance introduced above. Π_k is the space of all polynomials of degree $\leq k$.

Lemma 4.9 *Let $M = (M_{ij})_{i,j \in \mathcal{I}}$ be a symmetric positive definite matrix with $\sigma(M) \subset [a, b]$ $(0 < a \leq b)$. Then for $i \neq j$, the entries of M^{-1} satisfy the estimate*

$$|(M^{-1})_{ij}| \leq \hat{c}\, q^{\delta_{ij}} \quad \text{with } \hat{c} = \frac{(1 + \sqrt{r})^2}{2ar}, \; q = \frac{\sqrt{r} - 1}{\sqrt{r} + 1}, \; r = \frac{b}{a}. \tag{4.16}$$

Proof: For any polynomial $p \in \Pi_k$ with $k < \delta_{ij}$ we observe that $p(M)_{ij} = 0$. 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. [27, p. 33]) there is a polynomial $p_k \in \Pi_k$ so that

$$\|\mu^{-1} - p_k(\mu)\|_{\infty,[a,b]} \leq \hat{c}\, q^{k+1}$$

with q and $\hat{c}$ as in (4.16). Set $k := \delta_{ij} - 1$. The previous arguments show the final result:

$$|(M^{-1})_{ij}| = |(M^{-1})_{ij} - p_k(M)_{ij}| \leq \|M^{-1} - p_k(M)\|_2 \leq \hat{c}\, q^{k+1} = \hat{c}\, q^{\delta_{ij}}.$$

■

Now, we apply Lemma 4.9 to the mass matrix. Its spectrum is contained in $[a,b]$ with $a = \mu_{\min}$ and $b = \mu_{\max}$. Due to Lemma 4.7, the ratio $r = \mu_{\max}/\mu_{\min}$ is bounded independently of the mesh size, i.e., independently of the size of the matrix M. Together with $\delta_{ij} \geq 1 + d_{ij}/h_{\min}$ from Lemma 4.8 we obtain

Lemma 4.10 *Let $\mathcal{T}$ be shape regular and quasi-uniform with $h_{\min}$ being the minimal side length of the triangles $t \in \mathcal{T}$. Then the inverse of the mass matrix satisfies*

$$|(M^{-1})_{ij}| \leq C\, \|M^{-1}\|_2\, q^{d_{ij}/h} \quad \textit{for all } i \neq j \in I,$$

where $C = \frac{r-1}{2r}$ and $q = \frac{\sqrt{r}-1}{\sqrt{r}+1} \in (0,1)$ with $r = \mu_{\max}/\mu_{\min}$ are independent of the matrix size n.

Approximation by an $\mathcal{H}$-Matrix

While the global Frobenius norm is easily described by those of the submatrices, the situation is more involved for the spectral norm. Given the partition P (leaves of $T_{\mathcal{I}\times\mathcal{I}}$), we denote by P_ℓ the subset $\{t \times s \in P : level(t) = \ell\}$.

Lemma 4.11 *Let P be a partition described by a cluster tree of depth $L = \mathcal{O}(\log n)$. Then there is a constant C_{sp} such that for any matrix $M \in \mathbb{R}^{n\times n}$ the following inequality holds between the global and the blockwise spectral norms:*

$$\|M\|_2 \leq C_{sp} \sum_{\ell=0}^{L} \max_{t\times s\in P_\ell} \|M|_{\hat{t}\times\hat{s}}\|_2. \tag{4.17}$$

Proof: Exercise ■

Under the assumption of shape regularity and quasi-uniformity, we have the estimate

$$\mathrm{vol}(\Omega_i) \geq c_v h^d \tag{4.18}$$

as well as the inequalities (4.15) discussed above. The supports Ω_i may overlap. In accordance with the standard finite element discretisation we require that each triangle t belongs to the support of a bounded number of basis functions, i.e., there is a constant $c_M > 0$ so that

$$c_M\ \mathrm{vol}(t) \geq \sum_{i\in t} \mathrm{vol}(\Omega_i), \qquad t \in \mathcal{T}. \tag{4.19}$$

Theorem 4.12 *Assume (4.18), (4.15), (4.19) and define the partition P of the mass matrix M by the admissibility condition*

$$\mathrm{dist}(\Omega_t, \Omega_s) \geq \rho \max\{\mathrm{diam}(\Omega_t), \mathrm{diam}(\Omega_s)\} > 0 \quad \textit{or} \quad \min\{\#\hat{t}, \#\hat{s}\} = 1. \tag{4.20}$$

Then for any $\varepsilon > 0$, there is $N_{\mathcal{H}} \in \mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, k_\varepsilon)$ satisfying $\|M^{-1} - N_{\mathcal{H}}\|_2 \leq \varepsilon \|M^{-1}\|_2$ with $k_\varepsilon = \mathcal{O}(\log^d(\frac{L}{\varepsilon}))$ ($L = \mathcal{O}(\log n)$ from Lemma 4.11).

Proof: (a) We use the following explicit definition of $N_{\mathcal{H}} = N_{\mathcal{H}}(k)$ depending on $k \in \mathbb{N}$. Set

$$\begin{aligned} N_{\mathcal{H}}|_{\hat{t}\times\hat{s}} &:= M^{-1}|_{\hat{t}\times\hat{s}} \text{ if } \#\hat{t}\#\hat{s} \le k^2, \\ N_{\mathcal{H}}|_{\hat{t}\times\hat{s}} &:= 0 \quad \text{otherwise}, \qquad\qquad \text{for } t\times s \in P. \end{aligned}$$

In the first case we have $\operatorname{rank}(N_{\mathcal{H}}|_{\hat{t}\times\hat{s}}) \le k$, while in the second case $\operatorname{rank}(N_{\mathcal{H}}|_{\hat{t}\times\hat{s}}) = 0 \le k$. Therefore $N_{\mathcal{H}}$ belongs to $\mathcal{H}(P,k)$. We define the error matrix

$$E := M^{-1} - N_{\mathcal{H}}(k).$$

Due to Lemma 4.11, it remains to determine the spectral norms of $E|_{\hat{t}\times\hat{s}} = M^{-1}|_{\hat{t}\times\hat{s}}$ in the case of $\#\hat{t}\#\hat{s} > k^2$.

(b) For $\#\hat{t}\#\hat{s} > k^2$ and $i \in \hat{t}$, $j \in \hat{s}$, we want to estimate $E_{ij} = (M^{-1})_{ij}$. Condition (4.20) implies

$$d_{ij} = \operatorname{dist}(\Omega_i, \Omega_j) \ge \operatorname{dist}(\Omega_t, \Omega_s) \ge \rho \max\{\operatorname{diam}(\Omega_t), \operatorname{diam}(\Omega_s)\}.$$

We notice that $(\operatorname{diam}(\Omega_t))^d \ge \operatorname{vol}(\Omega_t) 2^d/\omega_d$ (ω_d: volume of the unit sphere), and from (4.19) and (4.18) we obtain that

$$\operatorname{vol}(\Omega_t) \ge c_M^{-1} \sum_{i\in\hat{t}} \operatorname{vol}(\Omega_i) \ge \frac{c_v}{c_M} h^d \#\hat{t}.$$

Altogether, $d_{ij} \ge C' h \sqrt[d]{\#\hat{t}}$ follows with C' expressed by ω_d, ρ, c_M, and c_v. Similarly, $d_{ij} \ge C' h \sqrt[d]{\#\hat{s}}$ holds. The combination yields $d_{ij}/h \ge C' \sqrt[2d]{\#\hat{t}\#\hat{s}}$. This proves

$$|E_{ij}| \le C \|M^{-1}\|_2 q^{C' \sqrt[2d]{\#\hat{t}\#\hat{s}}}.$$

(c) A trivial estimate of the spectral norm yields

$$\|E|_{\hat{t}\times\hat{s}}\|_2 \le \sqrt{\#\hat{t}\#\hat{s}} \max_{i\in\hat{t}, j\in\hat{s}} |E_{ij}| \le C \sqrt{\#\hat{t}\#\hat{s}} \|M^{-1}\|_2 q^{C' \sqrt[2d]{\#\hat{t}\#\hat{s}}}.$$

We simplify the right-hand side: for a suitable $C'' < C'$, the estimate $C\ell\, q^{C' \sqrt[d]{\ell}} \le q^{C'' \sqrt[d]{\ell}}$ holds for all $\ell \ge k_{\min}$ so that

$$\|E|_{\hat{t}\times\hat{s}}\|_2 \le \|M^{-1}\|_2 q^{C'' \sqrt[2d]{\#\hat{t}\#\hat{s}}} < \|M^{-1}\|_2 q^{C'' \sqrt[d]{k}}.$$

Lemma 4.11 implies $\|E\|_2 \le (L+1) C_* \|M^{-1}\|_2\, q^{C'' \sqrt[d]{k}}$ with $C_* = CC_{sp}$. Choose $k = k_\varepsilon \ge k_{\min}$ such that $(L+1)C_*\, q^{C'' \sqrt[d]{k}} \le \varepsilon$, i.e., $k_\varepsilon = \max\{k_{\min}, \mathcal{O}(\log^d(\frac{LC_*}{\varepsilon})\} = \mathcal{O}(\log^d(\frac{L}{\varepsilon}))$. ■

We summarise that the simple construction used in the proof yields an $\mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, k)$-approximation with $k = \mathcal{O}(\log^d(\frac{L}{\varepsilon}))$.

4.3.3 Connection between A^{-1} and B

The integral operator $\mathcal{G}$ from (4.13) leads to the Galerkin discretisation matrix $B = J^* L^{-1} J$ with entries

$$B_{ij} := \int_\Omega \int_\Omega \varphi_j(x) g(x,y) \varphi_i(y)\, dx dy \qquad (i,j \in \mathcal{I}),$$

where g is the Green function.

There are two FE error estimates which can be considered. The $L^2(\Omega)$-orthogonal projection is expressed by

$$Q_h := JM^{-1}J^* : L^2(\Omega) \to V_h, \quad \text{i.e.,} \quad (Q_h u, v_h)_{L^2} = (u, v_h)_{L^2} \quad \text{for all } u \in V \text{ and } v_h \in V_h.$$

The related error is described by

$$e_h^Q(u) := \|u - Q_h u\|_{L^2(\Omega)}.$$

On the other hand, the finite element approximation is connected with the Ritz projection

$$P_h = JA^{-1}J^*L : V \to V_h.$$

If $u \in V$ is the solution of the variational problem $a(u,v) = f(v)$ (cf. (4.11)), $u_h = P_h u$ is its finite element solution. The FE error is

$$e_h^P(u) := \|u - P_h u\|_{L^2(\Omega)}.$$

Since the $L^2(\Omega)$-orthogonal projection is the optimal one, i.e., $e_h^Q(u) \le e_h^P(u)$, we only need estimates of e_h^P. As proved in [1], the weakest form of the finite element convergence is described by

$$e_h^P(u) \le \varepsilon_h \|f\|_{L^2(\Omega)} \quad \text{for all } u = L^{-1}f,\ f \in L^2(\Omega), \tag{4.21}$$

where $\varepsilon_h \to 0$ as $h \to 0$. Only under further smoothness conditions on the coefficients c_{ij} (see (4.9)) and regularity assumptions, we can expect a better behaviour $\varepsilon_h = \mathcal{O}(h^\sigma)$ with $\sigma \in (0,2]$.

Lemma 4.13 *Let $c_{J,2}$ and ε_h be the quantities in (4.15) and (4.21). Then $\|MA^{-1}M - B\|_2 \le 2\,c_{J,2}^2\,\varepsilon_h$.*

Proof: Let $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ and $f_h = J\mathbf{x}$, $v_h = J\mathbf{y} \in V_h$. Then, using $B = J^*L^{-1}J$ and the projections from above, we have

$$\begin{aligned}
\langle (MA^{-1}M - B)\mathbf{x}, \mathbf{y}\rangle &= \langle (MA^{-1}M - J^*L^{-1}J)M^{-1}J^*f_h, M^{-1}J^*v_h\rangle \\
&= \big((JA^{-1}J^* - JM^{-1}J^*L^{-1}JM^{-1}J^*)f_h, v_h\big)_{L^2(\Omega)} \\
&= (P_hL^{-1}f_h - Q_hL^{-1}Q_hf_h, v_h)_{L^2(\Omega)} = (P_hL^{-1}f_h - Q_hL^{-1}f_h, v_h)_{L^2(\Omega)} \\
&= ([L^{-1}f_h - P_hL^{-1}f_h] - [L^{-1}f_h - Q_hL^{-1}f_h], v_h)_{L^2(\Omega)} \\
&\le \Big(e_h^P(L^{-1}f_h) + e_h^Q(L^{-1}f_h)\Big)\,\|v_h\|_{L^2(\Omega)} \le 2\,e_h^P(L^{-1}f_h)\|v_h\|_{L^2(\Omega)} \\
&\le 2\,\varepsilon_h\|f_h\|_{L^2(\Omega)}\|v_h\|_{L^2(\Omega)} \le 2\,c_{J,2}^2\,\varepsilon_h\|\mathbf{x}\|_h\|\mathbf{y}\|_h,
\end{aligned}$$

which proves $\|MA^{-1}M - B\|_2 \le 2\,c_{J,2}^2\,\varepsilon_h$. ■

Corollary 4.14 $\|A^{-1} - M^{-1}BM^{-1}\|_2 \le 2\,c_{J,1}^{-4}\,c_{J,2}^2\,\varepsilon_h$.

The further plan is as follows. In §4.3.5 we prove that B can be well approximated by an $\mathcal{H}$-matrix $B_\mathcal{H}$. From Theorem 4.12 we know that M^{-1} has an $\mathcal{H}$-matrix approximation $N_\mathcal{H}$. Due to Theorem 6.18, the product $N_\mathcal{H}B_\mathcal{H}N_\mathcal{H}$ yields an $\mathcal{H}$-matrix approximating $M^{-1}BM^{-1}$. Since additional errors of the order of the discretisation error ε_h are acceptable, approximations of $M^{-1}BM^{-1}$ are also good approximations of A^{-1} as shown in Corollary 4.14.

4.3.4 Green Functions

The Green function g is defined in $\Omega \times \Omega$. If ω_1 and ω_2 are disjoint, the restriction of g to $\omega_1 \times \omega_2$ is L-harmonic, i.e., $Lg = 0$. The subspace of L-harmonic functions will be considered in §4.3.4. First we give approximation results for general closed subspaces of $L^2(D)$.

Approximation by Finite Dimensional Subspaces

In the following lemmata $D \subset \mathbb{R}^d$ 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. The constant c_{appr} in (4.22) depends only on the spatial dimension d.

Lemma 4.15 *Let $D \subset \mathbb{R}^d$ be a convex domain and X a closed subspace of $L^2(D)$. Then for any $k \in \mathbb{N}$ there is a subspace $V_k \subset X$ satisfying $\dim V_k \le k$ so that*

$$\operatorname{dist}_{L^2(D)}(u, V_k) \le c_{appr} \frac{\operatorname{diam}(D)}{\sqrt[d]{k}} \|\nabla u\|_{L^2(D)} \qquad \text{for all } u \in X \cap H^1(D). \tag{4.22}$$

Proof: (a) First we assume $k = \ell^d$ and $D \subset Q = \{x \in \mathbb{R}^d : \|x - z\|_\infty < \frac{1}{2}\operatorname{diam}(D)\}$ for some $z \in \mathbb{R}^d$. We subdivide the cube Q uniformly into k subcubes Q_i, $i = 1, \dots, k$, and set $D_i = D \cap Q_i$, $i = 1, \dots, k$. Each of the sets D_i is convex with $\operatorname{diam}(D_i) \le \frac{\sqrt{d}}{\ell}\operatorname{diam}(D)$. Let

$$W_k = \{v \in L^2(D) : v \text{ is constant on } D_i \text{ for all } i = 1, \dots, k\}.$$

Then $\dim W_k \le k$ and according to Poincaré's inequality for $u \in H^1(D)$ it holds that

$$\int_{D_i} |u - \bar{u}_i|^2 dx \le \pi^{-2} \operatorname{diam}^2(D_i) \int_{D_i} |\nabla u|^2 dx,$$

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

$$\operatorname{dist}_{L^2(D)}(u, W_k) \le \|u - \bar{u}\|_{L^2(D)} \le \frac{\sqrt{d}}{\pi \ell} \operatorname{diam}(D) \, \|\nabla u\|_{L^2(D)}$$

for $\bar{u}$ defined by $\bar{u}|_{D_i} = \bar{u}_i$.

(b) 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

$$\operatorname{dist}_{L^2(D)}(u, W_k) \le c_{appr} \frac{\operatorname{diam}(D)}{\sqrt[d]{k}} \, \|\nabla u\|_{L^2(D)}$$

with the constant $c_{appr} := 2\sqrt{d}\, c_d$.

(c) Let $P : L^2(D) \to X$ be the $L^2(D)$-orthogonal projection onto X and $V_k = P(W_k)$. Keeping in mind that P has norm one and $u \in X$, the assertion follows from $\|u - P\bar{u}\|_{L^2(D)} = \|P(u - \bar{u})\|_{L^2(D)} \le \|u - \bar{u}\|_{L^2(D)}$. ■

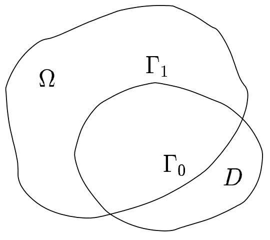
In the last proof we have restricted D_i to convex domains though Poincaré's inequality holds whenever the embedding $H^1(D_i) \hookrightarrow L^2(D_i)$ is compact. This is for example true if D_i fulfils a uniform cone condition. However, in this case it is not obvious how the constant depends on the geometry.

Space of L-Harmonic Functions

The Green function $g(x, \cdot)$ is a special example of an L-harmonic function in a subdomain $D_\Omega \subset \Omega$ (provided $x \notin D_\Omega$) with zero boundary values on $\partial\Omega \cap \overline{D_\Omega}$. The space X in Lemma 4.15 will be substituted by a function space $X(D) \subset L^2(D)$ which we define next. While the notation $X(D)$ will be used for different D, the underlying domain Ω is fixed.

Let D be a domain intersecting Ω: $D_\Omega := D \cap \Omega \ne \emptyset$. The boundary ∂D_Ω consists of two parts:

$$\Gamma_0(D) := D \cap \partial\Omega, \quad \Gamma_1(D) := \partial D_\Omega \backslash \Gamma_0(D) = \partial D \cap \overline{\Omega}. \tag{4.23}$$



$\Gamma_0 = \emptyset$ holds in the cases of $D \subset \Omega$ or $D \supset \Omega$. The former case may happen, whereas the latter is of no interest for us.

If D is not a subset of Ω, we require that outside of Ω functions $u \in X(D)$ are extended by zero. The functions $u \in X(D)$ are locally in $H^1(D)$ relative to $\Gamma_1(D)$ (notation: $u \in H^1_{rl,\Omega}(D)$) in the following sense:

$$H^1_{rl,\Omega}(D) := \{u \in L^2(D) : u|_{D\setminus\Omega} = 0,\ u \in H^1(K) \text{ for all } K \subset D \text{ with } \operatorname{dist}(K,\Gamma_1(D)) > 0\}. \tag{4.24}$$

The first condition is empty if $D \subset \Omega$.

The L-harmonicity[1] is required in the weak formulation of $Lu = 0$,

$$a(u,\varphi) = 0 \qquad \text{for all } \varphi \in C_0^\infty(D_\Omega) \qquad\qquad (D_\Omega = D \cap \Omega) \tag{4.25}$$

with $a(\cdot,\cdot)$ from (4.11). The final definition is

$$X(D) := \{u \in H^1_{rl,\Omega}(D) : u \text{ satisfies (4.25)}\}. \tag{4.26}$$

The Green function $g(x,\cdot)$ can be extended to D by zero. This extension is in $H^1(D)$ and hence in $X(D)$ if $x \in \Omega \setminus \overline{D}$.

Lemma 4.16 *The space $X(D)$ is closed in $L^2(D)$.*

The proof is postponed to the next subsection, since it needs Lemma 4.18. The closeness of $X(D)$ is necessary in order to use $X(D)$ as X in Lemma 4.15.

Remark 4.17 *Consider $X(D)$ and $X(D')$ for two domains $D' \subset D$ intersecting Ω.*

(a) For any $u \in X(D)$, the restriction $u|_{D'}$ belongs to $X(D')$; hence, in short notation, $X(D)|_{D'} = X(D')$. If $\operatorname{dist}(D',\Gamma_1(D)) > 0$, even $X(D)|_{D'} = X(D') \cap H^1(D')$ holds (cf. (4.24)).

(b) The relevant parts of D and D' are $D_\Omega = D \cap \Omega$, $D'_\Omega = D' \cap \Omega$ as well as $\Gamma_0(D)$ and $\Gamma_0(D')$. D_Ω and D'_Ω are the domains of L-harmonicity, whereas $\Gamma_0(D)$ and $\Gamma_0(D')$ describe the location of zero boundary values due to the zero extension outside. As long as $D = D'$ and $\Gamma_0 = \Gamma'_0$, differences in $D \setminus \overline{\Omega}$ and $D' \setminus \overline{\Omega}$ are irrelevant, since functions from $X(D)$ and $X(D')$ vanish in these parts anyway.

The Caccioppoli Inequality

The following lemma shows that any function $u \in X(D)$ allows to estimate $\|\nabla u\|_{L^2(K_\Omega)}$ for a domain $K \subset D$ not touching $\Gamma_1(D)$ by means of the weaker norm $\|u\|_{L^2(D_\Omega)}$. Note that K may contain parts of $\Gamma_0(D)$.

Lemma 4.18 *Let $X(D)$, $\Gamma_1(D)$ as in (4.26), (4.23), and $K \subset D$, $K_\Omega = K \cap \Omega$ with $\operatorname{dist}(K,\Gamma_1(D)) > 0$. Further, let $\kappa_C = \lambda_{\max}/\lambda_{\min}$ (cf. (4.12)). Then the so-called* Caccioppoli inequality *holds:*

$$\|\nabla u\|_{L^2(K_\Omega)} \le \frac{4\sqrt{\kappa_C}}{\operatorname{dist}(K,\Gamma_1(D))}\|u\|_{L^2(D_\Omega)} \qquad \textit{for all } u \in X(D). \tag{4.27}$$

Proof: Let $\eta \in C^1(D)$ satisfy $0 \le \eta \le 1$, $\eta = 1$ in K, $\eta = 0$ in a neighbourhood of $\Gamma_1(D)$ and $|\nabla\eta| \le 2/\delta$ in D_Ω, where we set $\delta = \operatorname{dist}(K,\Gamma_1(D))$. Since $K' := \operatorname{supp}(\eta) \subset D$ satisfies $\operatorname{dist}(K',\Gamma_1(D)) > 0$, (4.24) implies $u \in H^1(K')$. Hence, $\varphi := \eta^2 u \in H^1_0(D_\Omega)$ may be used as a test function in $a(u,\varphi) = 0$ due to the dense embedding of $C_0^\infty(D_\Omega)$ in $H^1_0(D_\Omega)$:

$$0 = \int_{D_\Omega} (\nabla u)^T C(x) \nabla(\eta^2 u)dx = 2\int_{D_\Omega} \eta u (\nabla u)^T C(x)(\nabla\eta)dx + \int_{D_\Omega} \eta^2 (\nabla u)^T C(x)(\nabla u)dx.$$

[1]To be precise, we need L^*-harmonicity, since the Green function $g(x,y)$ is L-harmonic w.r.t. x but L^*-harmonic w.r.t. y. However, since here L consists only of the principal part (4.9), L is self-adjoined. But notice that symmetry is not at all essential.

From (4.12) it follows that

$$\begin{aligned}\int_{D_\Omega} \eta^2 \left\|C^{1/2}(x)\nabla u\right\|^2 \mathrm{d}x &= \int_{D_\Omega} \eta^2 (\nabla u)^T C(x)(\nabla u)\mathrm{d}x = 2\left|\int_{D_\Omega} \eta u(\nabla u)^T C(x)(\nabla\eta)\mathrm{d}x\right| \\ &\le 2\int_{D_\Omega} \eta\,|u|\;\|C^{1/2}(x)\nabla\eta\|\;\|C^{1/2}(x)\nabla u\|\mathrm{d}x \\ &\le 4\,\frac{\sqrt{\lambda_{\max}}}{\delta}\int_{D_\Omega} |u|\left(\eta\|C^{1/2}(x)\nabla u\|\right)\mathrm{d}x \\ &\le 4\,\frac{\sqrt{\lambda_{\max}}}{\delta}\left(\int_{D_\Omega} \eta^2\|C^{1/2}(x)\nabla u\|^2\mathrm{d}x\right)^{1/2}\|u\|_{L^2(D_\Omega)},\end{aligned}$$

i.e., $\|\eta\, C^{1/2}(x)\nabla u\|_{L^2(D_\Omega)} \le 4\frac{\sqrt{\lambda_{\max}}}{\delta}\|u\|_{L^2(D_\Omega)}$. The estimation by

$$\|\nabla u\|_{L^2(K_\Omega)} \le \|\eta\nabla u\|_{L^2(D_\Omega)} \le \lambda_{\min}^{-1/2}\|\eta C^{1/2}(x)\nabla u\|_{L^2(D_\Omega)}$$

yields the assertion. ■

Remark 4.19 *Since $u = 0$ in $D\backslash\Omega$, we may write the norms in the inequality of Lemma 4.18 as $\|\nabla u\|_{L^2(K)}$ and $\|u\|_{L^2(D)}$ (i.e., K instead of K_Ω and D instead of D_Ω).*

Proof:(Lemma 4.16) Let $\{u_k\}_{k\in\mathbb{N}} \subset X(D)$ converge to u in $L^2(D)$. Let $K \subset D$ with $\mathrm{dist}(K, \Gamma_1(D)) > 0$. According to Remark 4.19, the sequence $\{\nabla u_k\}_{k\in\mathbb{N}}$ is bounded on K,

$$\|\nabla u_k\|_{L^2(K)} \le c\,\|u_k\|_{L^2(D)} \le C.$$

Due to the Banach-Alaoglu Theorem, a subsequence $\{u_{i_k}\}_{k\in\mathbb{N}}$ converges weakly in $H^1(K)$ to $\hat{u} \in H^1(K)$. Hence, for any $v \in L^2(K)$ we have $(u, v)_{L^2(K)} = \lim_{k\to\infty}(u_{i_k}, v)_{L^2(K)} = (\hat{u}, v)_{L^2(K)}$ proving $u = \hat{u} \in H^1(K)$. Since the functional $a(\cdot,\varphi)$ for $\varphi \in C_0^\infty(D_\Omega)$ is in $(H^1(K))'$, we see by the same argument that $a(u,\varphi) = 0$. Finally, $u_k|_{D\backslash\Omega} = 0$ leads to $u|_{D\backslash\Omega} = 0$. Hence, $u \in X(D)$ is shown. ■

Main Theorem

First we investigate how large the dimension of a finite dimensional subspace must be to approximate a function from $X(D)$ in a subdomain D_2 of D up to a certain error.

Lemma 4.20 *Let D, $\Gamma_1(D)$, D_Ω and $X(D)$ as before (cf. Lemma 4.18) and assume that $D_2 \subset D$ is a convex domain such that*

$$\mathrm{dist}(D_2, \partial D) \ge \rho\ \mathrm{diam}(D_2) > 0.$$

Then for any $\mu > 1$ there is a subspace $W \subset X(D_2)$ so that

$$\mathrm{dist}_{L^2(D_2)}(u, W) \le \frac{1}{\mu}\,\|u\|_{L^2(D_\Omega)} \qquad \textit{for all } u \in X(D) \tag{4.28}$$

and

$$\dim W \le c_\rho^d \lceil\log\mu\rceil^{d+1} + \lceil\log\mu\rceil, \quad c_\rho = 4ec_{appr}\sqrt{\kappa_C}\,\frac{1+2\rho}{\rho}. \tag{4.29}$$

Proof: (a) Consider $K(r) := \{x \in \mathbb{R}^d : \mathrm{dist}(x, D_2) \le r\}$ for $0 \le r \le \mathrm{dist}(D_2, \partial D)$. We conclude that $K(r)$ are again convex domains which are increasing with r: $K(r_1) \supseteq K(r_2)$ for $r_1 \ge r_2$. The smallest is $K(0) = D_2$, while $K(\mathrm{dist}(D_2, \partial D))$ is the largest one which is still in D. We remark that

$$\mathrm{dist}(K(r_2), \partial K(r_1)) = r_1 - r_2 \qquad \text{for } r_1 \ge r_2 \text{ and} \qquad \mathrm{diam}(K(r)) \le \mathrm{diam}(D_2) + 2r.$$

(b) Consider the sequence $r_0 > r_1 > \ldots > r_i = 0$ with $r_j := (1 - j/i)\,\mathrm{dist}(D_2, \partial D)$, where i is chosen later. Using $K(r)$ from Part (a) we set

$$D^j := K(r_j), \quad X^j := X(D^j) \qquad \text{(cf. (4.26))}$$

and notice that $D_2 = D^i \subset D^{i-1} \subset \ldots \subset D^0 \subset D$.

(c) Let $j \in \{1, \ldots, i\}$. Applying Lemma 4.18 (Remark 4.19) with (D^{j-1}, D^j) instead of (D, K), we obtain

$$\|\nabla v\|_{L^2(D^j)} \le \frac{4\sqrt{\kappa_C}}{\mathrm{dist}(D^j, \Gamma_1(D^{j-1}))}\,\|v\|_{L^2(D^{j-1})} \qquad \text{for all } v \in X^{j-1}$$

(we recall $\Gamma_1(D^{j-1}) = \partial D^{j-1} \cap \overline{\Omega}$). Because of $\mathrm{dist}(D^j, \Gamma_1(D^{j-1})) \ge \mathrm{dist}(D^j, \partial D^{j-1}) = r_{j-1} - r_j = r_0/i$ (see Part (a)), the resulting estimate is

$$\|\nabla v\|_{L^2(D^j)} \le \frac{4i\sqrt{\kappa_C}}{r_0}\,\|v\|_{L^2(D^{j-1})} \qquad \text{for all } v \in X^{j-1}. \tag{4.30}$$

(d) Apply Lemma 4.15 with D^j instead of D and with the choice $k := \lceil (\beta i)^d \rceil$, where the factor β will be adjusted later. Then this Lemma ensures that there is a subspace $V_j \subset X^j$ satisfying $\dim V_j \le k$ and

$$\mathrm{dist}_{L^2(D^j)}(v, V_j) \le c_{appr}\,\frac{\mathrm{diam}(D^j)}{\sqrt[d]{k}}\,\|\nabla v\|_{L^2(D^j)} \qquad \text{for all } v \in X^j \cap H^1(D^j).$$

Using $\sqrt[d]{k} \ge \beta i$ and $\mathrm{diam}(D^j) = \mathrm{diam}(D_2) + 2r_j \le \mathrm{diam}(D_2) + 2r_0$ (see Part (a)), we arrive at

$$\mathrm{dist}_{L^2(D^j)}(v, V_j) \le c_{appr}\,\frac{\mathrm{diam}(D_2) + 2r_0}{\beta i}\,\|\nabla v\|_{L^2(D^j)} \qquad \text{for all } v \in X^j \cap H^1(D^j). \tag{4.31}$$

Since any $v \in X^{j-1}$ also belongs to $X^j \cap H^1(D^j)$, the estimates (4.30), (4.31) together with $r_0 \ge \rho\,\mathrm{diam}(D_2)$ may be combined to

$$\mathrm{dist}_{L^2(D^j)}(v, V_j) \le \frac{1+2\rho}{\rho}\,\frac{4c_{appr}\sqrt{\kappa_C}}{\beta}\,\|v\|_{L^2(D^{j-1})} \qquad \text{for all } v \in X^{j-1}. \tag{4.32}$$

In particular, the factor $\frac{1+2\rho}{\rho}\frac{4c_{appr}\sqrt{\kappa_C}}{\beta}$ becomes $\mu^{-1/i}$ for the choice

$$\beta := \beta_0 M^{1/i} \quad \text{with } \beta_0 := 4c_{appr}\sqrt{\kappa_C}\,\frac{1+2\rho}{\rho}. \tag{4.33}$$

(e) For any given $u =: v_0 \in X^0$, (4.32) and (4.33) lead to $v_0|_{D^1} = u_1 + v_1$ with $u_1 \in V_1$ and

$$\|v_1\|_{L^2(D^1)} \le \mu^{-1/i}\,\|v_0\|_{L^2(D^0)}.$$

Consequently, v_1 belongs to X^1. Similarly, for all $j = 1, \ldots, i$ we are able to find an approximant $u_j \in V_j$ so that $v_{j-1}|_{D^j} = u_j + v_j$ and $\|v_j\|_{L^2(D^j)} \le \mu^{-1/i}\,\|v_{j-1}\|_{L^2(D^{j-1})}$. Hence, the subspace

$$W := \mathrm{span}\{V_j|_{D_2} : j = 1, \ldots, i\}$$

using the restrictions of V_j to the smallest domain $D_2 = D^i$ contains $u_j|_{D_2} \in V_j|_{D_2} \subset W$. Therefore, $v_0 = v_i + \sum_{j=1}^{i} u_j$ leads to

$$\mathrm{dist}_{L^2(D_2)}(v_0, W) \le \|v_i\|_{L^2(D_2)} \le \left(\mu^{-1/i}\right)^i \|v_0\|_{L^2(D^0)} \le \mu^{-1}\|u\|_{L^2(D_\Omega)},$$

where the last inequality is due to $D^0 \subset D$ and $u|_{D\setminus\Omega} = 0$.

(f) The dimension of W is bounded by $\sum_{j=1}^{i} \dim V_j = i\lceil (\beta i)^d \rceil \le i + \beta^d i^{d+1}$. The choice $i := \lceil \log \mu \rceil$ yields

$$\dim W \le \lceil \log \mu \rceil + \beta_0^d e^d \lceil \log \mu \rceil^{d+1}$$

because of $\beta = \beta_0 \mu^{1/i} \le \beta_0 e$. Together with $c_\rho = \beta_0 e$, we obtain the final result. ■

Remark 4.21 *(a) Setting $\mu = \exp(m)$, the dimension of W is bounded by $c_\rho^d \lceil m \rceil^{d+1} + \lceil m \rceil \sim c_\rho^d m^{d+1}$. On the other hand, if a dimension $K = \dim W$ is given, the possible improvement factor $\frac{1}{\mu} = \exp(-m)$ is described by $m \gtrsim (c_\rho K)^{1/(d+1)}/c_\rho$.*

(b) The factor $\frac{1+2\rho}{\rho}$ in (4.29) shows that ρ should be of order $\mathcal{O}(1)$, e.g., $\operatorname{dist}(D_2, \partial D) \geq \operatorname{diam}(D_2)$ is a reasonable choice.

Next we consider the Green functions $g(x, \cdot)$ with $x \in D_1 \subset \Omega$, which are L-harmonic in $\Omega \setminus \overline{D_1}$. Note that its approximant $g_k(x, \cdot)$ from the following theorem is of the desired form (4.14).

Theorem 4.22 *Let $D_1 \subset \Omega$ and D_2 with $D_2 \cap \Omega \neq \emptyset$ be two domains such that D_2 is convex and*

$$\operatorname{dist}(D_1, D_2) \geq \rho \operatorname{diam}(D_2) > 0.$$

Then for any $\varepsilon \in (0,1)$ there is a separable approximation

$$g_k(x,y) = \sum_{i=1}^{k} u_i(x) v_i(y) \quad \text{with } k \leq k_\varepsilon = c_{\rho/2}^d \lceil \log \frac{1}{\varepsilon} \rceil^{d+1} + \lceil \log \frac{1}{\varepsilon} \rceil,$$

where c_ρ is defined in (4.29), so that

$$\|g(x,\cdot) - g_k(x,\cdot)\|_{L^2(D_2)} \leq \varepsilon \|g(x,\cdot)\|_{L^2(D_\Omega)} \qquad \text{for all } x \in D_1, \tag{4.34}$$

where $D := \{y \in \mathbb{R}^d : \operatorname{dist}(y, D_2) \leq \frac{\rho}{2} \operatorname{diam}(D_2)\}$ and $D_\Omega := D \cap \Omega$.

Proof: Note that the right-hand side $\|g(x,\cdot)\|_{L^2(D_\Omega)}$ does not contain the singularity of g because of $\operatorname{dist}(D_\Omega, D_1) \geq \operatorname{dist}(D, D_1) = \operatorname{dist}(D_2, D_1) - \frac{\rho}{2}\operatorname{diam}(D_2) \geq \frac{\rho}{2}\operatorname{diam}(D_2) > 0$.

Since $\operatorname{dist}(D_2, \partial D) = \frac{\rho}{2}\operatorname{diam}(D_2)$, we can apply Lemma 4.20 with $M = \varepsilon^{-1}$ and ρ replaced by $\rho/2$. Let $\{v_1, \ldots, v_k\}$ be a basis of the subspace $W \subset X(D_2)$ with $k = \dim W \leq c_{\rho/2}^d \lceil \log \frac{1}{\varepsilon} \rceil^{d+1} + \lceil \log \frac{1}{\varepsilon} \rceil$ according to Lemma 4.20.

For any $x \in D_1$, the function $g_x := g(x, \cdot)$ is in $X(D)$. By means of (4.28), $g_x = \hat{g}_x + r_x$ holds with $\hat{g}_x \in W$ and $\|r_x\|_{L^2(D_2)} \leq \varepsilon \|g_x\|_{L^2(D_\Omega)}$. Expressing $\hat{g}_x$ by means of the basis, we obtain

$$\hat{g}_x = \sum_{i=1}^{k} u_i(x) v_i$$

with coefficients $u_i(x)$ depending on the index x. Since x varies in D_1, the u_i are functions defined on D_1. The function $g_k(x,y) := \sum_{i=1}^{k} u_i(x) v_i(y)$ satisfies estimate (4.34). ■

Remark 4.23 *Without loss of generality, we may choose $\{v_1, \ldots, v_k\}$ as an* orthogonal *basis of W. Then the coefficients $u_i(x)$ in the latter expansion equal $(g(x,\cdot), v_i)_{L^2(D_2 \cap \Omega)}$ showing that the u_i's satisfy $Lu_i = v_i$ with homogeneous Dirichlet boundary conditions. In particular, u_i is L-harmonic in $\Omega \setminus D_2$. Note that the u_i's do not depend on D_1.*

For later use, we add a trivial remark.

Remark 4.24 *Assume (4.34) and $E \subset D_1$. Then $\|g - g_k\|_{L^2(E \times D_2)} \leq \varepsilon \|g\|_{L^2(E \times D_\Omega)}$.*

Theorem 4.22 can be easily adjusted to fundamental solutions s,

$$L_x S(x,y) = \delta(x-y) \qquad \text{for all } x, y \in \mathbb{R}^d,$$

which play a central role for example in boundary element methods (BEM). The following corollary guarantees that we are able to treat BEM matrices by $\mathcal{H}$-matrices.

Corollary 4.25 *Assume that a fundamental solution S exists for L. Let $D_1, D_2 \subset \mathbb{R}^d$ be two domains with D_2 being convex and*

$$\operatorname{dist}(D_1, D_2) \geq \rho \operatorname{diam}(D_2) > 0.$$

Then for $\varepsilon > 0$ there is $S_k(x,y) = \sum_{i=1}^k u_i(x)v_i(y)$ with $k \leq k_\varepsilon = c_{\rho/2}^d \lceil \log \frac{1}{\varepsilon} \rceil^{d+1} + \lceil \log \frac{1}{\varepsilon} \rceil$, where c_ρ is defined in (4.29), so that

$$\|S(x,\cdot) - S_k(x,\cdot)\|_{L^2(D_2)} \leq \varepsilon \|S(x,\cdot)\|_{L^2(D)} \qquad \textit{for all } x \in D_1,$$

where $D := \{x \in \mathbb{R}^d : \operatorname{dist}(x, D_2) \leq \frac{1}{2} \operatorname{dist}(D_1, D_2)\}$.

4.3.5 FEM Matrices

Theorem 4.26 *Assume (4.20) and let Ω_s be convex for all $s \in T_{\mathcal{I}}$. Let $P := \mathcal{L}(T_{\mathcal{I}\times\mathcal{I}})$. For any $\varepsilon \in (0,1)$, let $k_\varepsilon \in \mathbb{N}$ ($k_\varepsilon \sim \mathcal{O}(\log^{d+1}(\frac{1}{\varepsilon}))$) be the dimension bound from Theorem 4.22. Then for $k \geq k_\varepsilon$ there is $B_{\mathcal{H}} \in \mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, k)$ such that the spectral norm of the difference is bounded by*

$$\|B - B_{\mathcal{H}}\|_2 \leq \varepsilon \frac{c(\kappa_C, \rho, \operatorname{diam}(\Omega))}{\lambda_{\min}} L, \tag{4.35}$$

where $c(\kappa_C, \rho, \Omega)$ is a function depending on $\kappa_C = \lambda_{\max}/\lambda_{\min}$, ρ from (4.20) and $\operatorname{diam}(\Omega)$. *$L = \mathcal{O}(\log n)$ is the maximal level from Lemma 4.11.*

Proof: (a) Let $b := t \times s \in P$ be admissible such that the first inequality of (4.20) holds. Apply Theorem 4.22 with $D_1 = \Omega_t$, $D_2 = \Omega_s$, and $D_\Omega := \{x \in \Omega : \operatorname{dist}(x, \Omega_s) \leq \frac{\rho}{2} \operatorname{diam}(\Omega_s)\}$. According to Remark 4.24 there is $\tilde{g}^b(x,y) = \sum_{i=1}^{k_\varepsilon} u_i^b(x) v_i^b(y)$ such that

$$\|g - \tilde{g}^b\|_{L^2(\Omega_t \times \Omega_s)} \leq \varepsilon \|g\|_{L^2(\Omega_t \times D_\Omega)}.$$

Let the functions u_i^b and v_i^b of $\tilde{g}^b$ be extended to Ω by zero. We define the integral operator

$$K_b \varphi = \int_\Omega \tilde{g}^b(\cdot, y) \varphi(y)\, dy \qquad \text{for } \operatorname{supp} \varphi \subset \overline{\Omega}$$

and set $B_{\mathcal{H}}|_b = (J^* K_b J)|_b$ for all blocks b. The rank of $B_{\mathcal{H}}|_b$ is bounded by k_ε since each term $u_i^b(x) v_i^b(y)$ in $\tilde{g}^b$ produces one rank-1 matrix in $(J^* K_b J)|_b$.

In the inadmissible blocks b we use the *exact* Green function, i.e., $\tilde{g}^b := g$.

(b) Consider an admissible block $b = t \times s \in P$. Choose any vectors $\mathbf{x} = (x_j)_{j \in s}$, $\mathbf{y} = (y_i)_{i \in t}$ and set $u = J\mathbf{x} = \sum_{j \in s} x_j \varphi_j$ and $v = J\mathbf{y}$. To see that $B_{\mathcal{H}}|_b$ approximates the block $B|_b$, remember the representation $(L^{-1}\varphi)(x) = \int_\Omega g(x,y)\varphi(y)\, dy$ of L^{-1} and use (4.15). The estimate

$$\begin{aligned}
|\langle (B|_b - B_{\mathcal{H}}|_b)\mathbf{y}, \mathbf{x}\rangle| &= |\langle J^*(L^{-1} - K_b) J\mathbf{y}, \mathbf{x}\rangle| = |((L^{-1} - K_b)v, u)_{L^2}| \\
&\leq \|g - \tilde{g}^b\|_{L^2(\Omega_t \times \Omega_s)} \|u\|_{L^2(\Omega_s)} \|v\|_{L^2(\Omega_t)} \\
&\leq \varepsilon \|g\|_{L^2(\Omega_t \times D_\Omega)} \|u\|_{L^2(\Omega)} \|v\|_{L^2(\Omega)} \\
&\leq \varepsilon\, c_{J,2}^2 \|g\|_{L^2(\Omega_t \times D_\Omega)} \|\mathbf{x}\| \|\mathbf{y}\|
\end{aligned}$$

proves $\|B|_b - B_{\mathcal{H}}|_b\|_2 \leq \varepsilon\, c_{J,2}^2 \|g\|_{L^2(\Omega_t \times D_\Omega)}$ for the spectral norm.

Although $g(\cdot, y) \in W^{1,1}(\Omega)$ for all $y \in \Omega$, $g(\cdot,\cdot)$ does not belong to $L^2(\Omega \times \Omega)$ as soon as $d \geq 4$. From [20]

$$|g(x,y)| \leq \frac{c(d, \kappa_C)}{\lambda_{\min}} |x - y|^{2-d} \tag{4.36}$$

it can be seen that $\|g\|_{L^2(\Omega_t \times D_\Omega)}$ may increase when the sets Ω_t, D_Ω are approaching each other. The construction of Ω_t ensures

$$\delta := \operatorname{dist}(\Omega_t, D_\Omega) = \frac{1}{2} \operatorname{dist}(\Omega_t, \Omega_s) \geq \frac{\rho}{2} \operatorname{diam}(\Omega_t)$$

as well as $\delta \geq \frac{\rho}{2}\operatorname{diam}(\Omega_s)$. Hence (4.36) implies

$$\|g\|_{L^2(\Omega_t\times D_\Omega)} \leq \frac{c(d,\kappa_C)}{\lambda_{\min}}\delta^{2-d}\sqrt{\operatorname{vol}(\Omega_t)\operatorname{vol}(D_\Omega)}.$$

Using $\operatorname{vol}(D_\Omega) \leq \omega_d(\frac{1}{2}\operatorname{diam}(D_\Omega))^d \leq \omega_d(1+1/\rho)^d\delta^d$ and $\operatorname{vol}(\Omega_t) \leq \omega_d(\delta/\rho)^d$ with $\omega_d = \operatorname{vol}(B_1(0))$, we see that

$$\|g\|_{L^2(\Omega_t\times D_\Omega)} \leq C_\rho\frac{c(d,\kappa_C)}{\lambda_{\min}}\delta^2 \quad \text{with } C_\rho := \omega_d\,\frac{(\rho+1)^{d/2}}{\rho^d}.$$

The rough estimate $\delta \leq \operatorname{diam}(\Omega) = \mathcal{O}(1)$ together with Lemma 4.11 yields (4.35). ■

Corollary 4.27 *Assume that each (possibly non-convex) set Ω_s has a convex superset Y_s (e.g., bounding box) satisfying the admissibility condition. Then Theorem 4.26 remains true for Ω_t, Ω_s.*

Proof: Apply Theorem 4.26 to Y_t and Y_s. ■

Take $B_\mathcal{H} \in \mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, k_B)$ from Theorem 4.26 and let $N_\mathcal{H} \in \mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, k_N)$ be the $\mathcal{H}$-matrix approximations of M^{-1} (cf. Theorem 4.12). The product $C_\mathcal{H} := N_\mathcal{H}B_\mathcal{H}N_\mathcal{H}$ belongs to $\mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, k)$ with $k := C_{\text{id}}^2C_{\text{sp}}^2(L+1)^2\max\{k_B,k_N\}$.

The estimation of the spectral norm of

$$M^{-1}BM^{-1} - N_\mathcal{H}B_\mathcal{H}N_\mathcal{H} = (M^{-1}-N_\mathcal{H})BM^{-1} + N_\mathcal{H}(B-B_\mathcal{H})M^{-1} + N_\mathcal{H}B_\mathcal{H}(M^{-1}-N_\mathcal{H})$$

by

$$\|M^{-1}-N_\mathcal{H}\|_2(\|B\|_2\|M^{-1}\|_2 + \|N_\mathcal{H}\|_2\|B_\mathcal{H}\|_2) + \|N_\mathcal{H}\|_2\|M^{-1}\|_2\|B-B_\mathcal{H}\|_2$$

is obvious. Let $\varepsilon_N := \|M^{-1}-N_\mathcal{H}\|_2$, $\varepsilon_B := \|B-B_\mathcal{H}\|_2$. Since $\varepsilon_N \leq \|M^{-1}\|_2$, $\varepsilon_B \leq \|B\|_2$ and $\|B\|_2, \|M^{-1}\|_2 = \mathcal{O}(1)$, we obtain

$$\|M^{-1}BM^{-1} - N_\mathcal{H}B_\mathcal{H}N_\mathcal{H}\|_2 \leq C_{II}(\varepsilon_N + \varepsilon_B). \tag{4.37}$$

The combination of Corollary 4.14 and (4.37) yields

$$\|A^{-1} - N_\mathcal{H}B_\mathcal{H}N_\mathcal{H}\|_2 \leq C_I\,\varepsilon_h + C_{II}(\varepsilon_N+\varepsilon_B),$$

where $C_I = 2\,c_{J,1}^{-4}\,c_{J,2}^2$. For simplicity we set

$$k_N := k_B := \max\{\mathcal{O}(\log^{d+1}(\frac{LC_1}{\delta})), \mathcal{O}(\log^d(\frac{L\|M^{-1}\|_2}{\delta}))\}$$

with $C_1 = \frac{c(\kappa_C,\rho,\operatorname{diam}(\Omega))}{\lambda_{\min}}$ and $\delta = C_I\,\theta\varepsilon_h/(2C_{II})$, where the constants in the $\mathcal{O}(\cdot)$ expressions are detailed in Theorem 4.26 and Theorem 4.12, while $\theta \in (0,1)$. Then,

$$\|A^{-1} - N_\mathcal{H}B_\mathcal{H}N_\mathcal{H}\|_2 \leq C_I\,(1+\theta)\varepsilon_h \tag{4.38}$$

shows that the already existing finite element error $C_I\,\varepsilon_h$ is only slightly increased. The corresponding ranks $k_B = k_N$ behave asymptotically like

$$k_B = k_N = \mathcal{O}(\log^{d+1}(\frac{L}{\varepsilon_h})).$$

The resulting rank for $C_\mathcal{H} = N_\mathcal{H}B_\mathcal{H}N_\mathcal{H}$ is bounded by $k_C = C_{\text{id}}^2C_{\text{sp}}^2(L+1)^2k_B$. Thus, $C_\mathcal{H}$ approximates A^{-1} as described in (4.38) and belongs to $\mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, k)$. This result is summarised in Part (a) of

Theorem 4.28 *(a) Let $\varepsilon_h > 0$ be the finite element error from (4.21). $L = \mathcal{O}(\log n)$ is the depth of the cluster tree (see Lemma 4.11). Then there are constants C' and C'' defining $k_C := C' L^2 \log^{d+1}(\frac{LC''}{\varepsilon_h})$ and there is an $\mathcal{H}$-matrix $C_{\mathcal{H}} \in \mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, k_C)$ such that*

$$\|A^{-1} - C_{\mathcal{H}}\|_2 \le C_I\,(1+\theta)\varepsilon_h. \tag{4.39}$$

(b) If $\varepsilon_h = \mathcal{O}(h^\beta)$ according to $e_h^P(u) \le c_E h^\beta \|f\|_{L^2(\Omega)}$ for all $u = L^{-1}f,\ f \in L^2(\Omega)$, with some $\beta > 0$, then $k_C = \mathcal{O}(\log^{d+3}(n))$ holds.

Proof: As $h^{-1} = O(n^{1/d})$, the asymptotic behaviour of $\log(\frac{LC''}{\varepsilon_h}) = \log(L) + \mathit{const} + \log(n^{\beta/d})$ is $\mathcal{O}(\log n)$. This proves Part (b). ■

Since $\lambda_{\min}$ in (4.12) is of size $\mathcal{O}(1)$ (without loss of generality, we may scale the problem so that $\lambda_{\min} = 1$), also $\|A^{-1}\|_2 = \mathcal{O}(1)$ holds. Hence, the absolute error (4.39) may be changed into a relative one: $\|A^{-1} - C_{\mathcal{H}}\|_2 \le C_I^*\,\|A^{-1}\|_2(1+\theta)\varepsilon_h$ with another constant C_I^*.

4.4 Implementation

The matrix G defined by (4.7) is sparse, so we should store it in a suitable format.

4.4.1 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 4.29 (`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.*

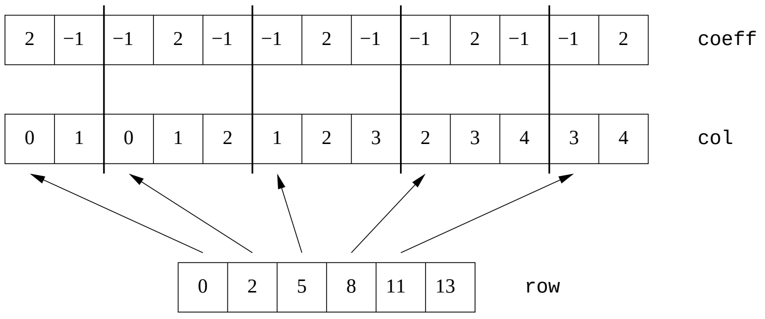
The columns indices for all non-zero entries are stored in the array `col`*, and the array* `coeff` *contains the corresponding coefficients. The array* `row` *contains the start and end indices of the parts of the* `col` *and* `coeff` *arrays corresponding to a certain row: For each row* `i`*, the entry* `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 4.30 (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 discretizations, 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 4.3, 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 4.31 (`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*

```
psparsefactory
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 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 2.11: we use piecewise linear basis functions, i.e., an entry G_{ij} of the stiffness matrix (4.7) 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 2.11 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.

4.4.2 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 (4.7). As in Example 2.11, we consider only domains that correspond to compatible triangulations, i.e., there is a family $(\Delta_i)_{i=0}^{T-1}$ of triangles such that $\overline{\Omega} = \overline{\bigcup_{i=0}^{T-1} \Delta_i}$ holds and that two different triangles Δ_i, Δ_j are either disjoint, share a common vertex or a common edge.

The discrete space V_n of piecewise linear functions on the triangulation $(\Delta_\iota)_{\iota=0}^{T-1}$ is defined by

$$V_n := \{u \in C(\Omega) \ : \ u|_{\Delta_\iota} \text{ is affine for all } \iota \in \{0, \ldots, T-1\} \text{ and } u|_\Gamma = 0\} \subseteq H_0^1(\Omega)$$

Let $u \in V_n$. Since u is affine on each Δ_ι, its restriction $u|_{\Delta_\iota}$ it is completely defined by the values in the vertices of Δ_ι. Since u is continuous, it is completely 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 $(v_i)_{i=0}^{T-1}$ and define the basis functions $(\varphi_i)_{i=0}^{T-1}$ by

$$\varphi_i(v_j) = \delta_{ij}$$

for all $j \in \{0, \ldots, T-1\}$. Due to (4.7), the entry G_{ij} is given by

$$G_{ij} = \int_\Omega \langle \nabla\varphi_i(x), \nabla\varphi_j(x)\rangle \, \mathrm{d}x = \sum_{\iota=0}^{T-1} \int_{\Delta_\iota} \langle \nabla\varphi_i(x), \nabla\varphi_j(x)\rangle \, \mathrm{d}x,$$

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));
    }
```

Since switching between triangles usually requires more operations than switching between basis functions, a different ordering of the loops is more efficient:

```
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 (4.7), we need the gradients of the basis functions. For piecewise linear basis functions, their computation is especially simple: Let $i, j, k \in \{0, \dots, n-1\}$ be such that v_i, v_j, v_k are the vertices of a triangle Δ_ι in the triangulation. The function

$$\varphi_{i,\Delta_\iota}(x) := \frac{\det(x - v_j, v_k - v_j)}{\det(v_i - v_j, v_k - v_j)}$$

if affine and obviously satisfies $\varphi_{i,\Delta_\iota}(v_i) = 1$, $\varphi_{i,\Delta_\iota}(v_j) = \varphi_{i,\Delta_\iota}(v_k) = 0$. This implies $\varphi_i|_{\Delta_\iota} = \varphi_{i,\Delta_\iota}$, so 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 find the complete algorithm for matrix assembly 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);
        }
      }
    }
  }
```

4.5 Exercises

4.5.1 Theory

Exercise 11 (Inversion of Banded Matrices) *Let $n = 2^p$ and $M \in \mathbb{R}^{n\times n}$ be a matrix with k upper and lower diagonals ($M_{ij} = 0$ for $|i-j| > k$). Let all principal matrices of M be regular.*

Prove that

$$M^{-1} \in \mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, k)$$

for the block cluster tree $T_{\mathcal{I}\times\mathcal{I}}$ from Section 4.1.2.

Exercise 12 ($\mathcal{H}(T,k)$ is Closed) *Let $T_{\mathcal{I}\times\mathcal{I}}$ be some block cluster tree and $k \in \mathbb{N}$. Prove that the set $\mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, k)$ is closed, i.e., for all $M \in \mathbb{R}^{\mathcal{I}\times\mathcal{I}}$ and all $(M_i)_{i\in\mathbb{N}} \subset \mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, k)$ holds*

$$\lim_{i\to\infty} \|M - M_i\| = 0 \quad \Rightarrow \quad M \in \mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, k)$$

Exercise 13 (Local Inversion Formula) *Let $M, P \in \mathbb{R}^{n\times n}$. Let M be regular. Prove that for any small enough $\varepsilon > 0$ holds*

$$(M + \varepsilon P)^{-1} = M^{-1} \sum_{\nu=0}^{\infty} (-\varepsilon P M^{-1})^{\nu} = M^{-1} + \mathcal{O}(\varepsilon).$$

Exercise 14 (Inversion of Tridiagonal Matrices) *In Lemma 4.1 and Exercise 11 we demanded the regularity of all principal matrices of M, because this is needed for the existence of an LU-decomposition. Now let $n = 2^p$ and $M \in \mathbb{R}^{n\times n}$ be a tridiagonal regular matrix whose principal matrices are not necessarily regular. Prove*

$$M^{-1} \in \mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, 1)$$

for the block cluster tree $T_{\mathcal{I}\times\mathcal{I}}$ from Section 4.1.2.

4.5.2 Practice

Exercise 15 (Anisotropic problem) *Let $M \in \mathbb{R}^{2\times 2}$ be a symmetric positive definite matrix. Write a function* `anisotropic_grid2d` *that builds a* `sparsematrix` *corresponding to the anisotropic equation*

$$-\langle \nabla, M\nabla\mathcal{U}\rangle(x) = -\sum_{i=1}^{2}\sum_{j=1}^{2} \partial_i M_{ij} \partial_j \mathcal{U}(x) = \mathcal{F}(x).$$

Hint: *Applying partial integration to the differential operator on the left side leads to matrix entries*

$$G_{ij} = \int_{\Omega} \langle M\nabla\varphi_i(x), \nabla\varphi_j(x)\rangle \, \mathrm{d}x.$$

Exercise 16 (Convection-diffusion problem) *Let $\epsilon \in \mathbb{R}_{>0}$ and $c \in \mathbb{R}^2$. Write a function* `convdiff_grid2d` *that builds a* `sparsematrix` *corresponding to the convection-diffusion equation*

$$-\epsilon\Delta\mathcal{U}(x) + \langle c, \nabla\mathcal{U}(x)\rangle = \mathcal{F}(x).$$

Hint: *Since partial integration does not apply to the lower-order part of the differential operator on the left side, it will be necessary to integrate a basis function φ_i. This can be done by the simple midpoint quadrature rule.*

Chapter 5

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 6. The basic idea for the $\mathcal{H}$-matrix arithmetics is formulated in [23] and a general approach is contained in [14] (german) and [16] (english).

5.1 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.

Definition 5.1 ($\mathcal{R}(k)$-Matrices) *Let $n, m, k \in \mathbb{N}$. We define the set of $n \times m$ matrices of rank at most k by*

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

Implementation 5.2 (`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;
};
```

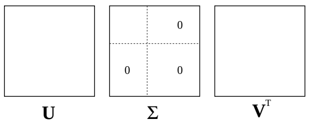
The description is the same as in the previous Implementation 1.12, except that upon initialisation (`new_rkmatrix`*) the current rank is* $\mathtt{kt} = 0$ *(the matrix is initialised to zero).*

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 4.3 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.

5.1.1 Reduced Singular Value Decomposition (rSVD)

Definition 5.3 (SVD and rSVD) *Let $M \in \mathcal{R}(k,n,m)$. A singular value decomposition (SVD) of M is a factorisation of the form*

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



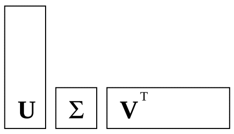
with unitary matrices $U \in \mathbb{R}^{n\times n}$, $V \in \mathbb{R}^{m\times m}$ and a diagonal matrix $\Sigma \in \mathbb{R}^{n\times m}$, where the diagonal entries are

$$\Sigma_{11} \geq \Sigma_{22} \geq \ldots \geq \Sigma_{kk} \quad \geq \quad \Sigma_{k+1,k+1} = \ldots = \Sigma_{\min\{n,m\},\min\{n,m\}} = 0.$$

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 5.4

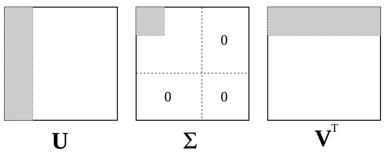
1. *A (reduced) SVD is not unique.*
2. *If the singular values of M are all different then the reduced singular value decomposition is unique up to scaling of the columns of U, V by -1.*
3. *A SVD of M yields a rSVD by discarding the columns $> k$ of U, V and the columns and rows $> k$ of Σ.*

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 [9]). 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.

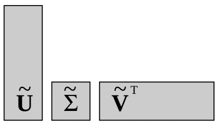
Lemma 5.5 (Best Approximation with Fixed Rank) *Let*

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



be a SVD of $M \in \mathbb{R}^{n\times m}$. Let

$$\tilde{M} := \tilde{U}\tilde{\Sigma}\tilde{V}^T$$



with matrices

$$\tilde{U} := U|_{n\times k}, \quad \tilde{\Sigma} := \operatorname{diag}(\Sigma_{1,1},\dots,\Sigma_{k,k}), \quad \tilde{V} := V|_{m\times k}.$$

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 [9]. 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`$\to$ complexity $\mathcal{O}((n+m)k^2)$
2. Compute a rSVD of $R_A R_B^T = U'\Sigma V'$. `dgesvd`$\to$ complexity $\mathcal{O}(k^3)$
3. Compute $U := Q_A U'$, $V := Q_B V'$. `dgemm`$\to$ complexity $\mathcal{O}((n+m)k^2)$

Implementation 5.6 (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 5.5 the rSVD representation can be used to derive a best approximation with fixed rank.

Definition 5.7 (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.

Definition 5.8 (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". $\mathcal{T}_\varepsilon^{\mathrm{abs}}$ is the respective truncation operator with absolute truncation error ε.

From Lemma 5.5 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.

5.1.2 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 5.9 (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 5.10 (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 5.11 (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 5.12 (Formatted `rkmatrix` Addition) *The structure* `truncation_control` *contains the information about the allowed relative truncation error* `rel_eps`*, the allowed 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*

$$k := \min\{\tilde{k} \in \mathbb{N}_0 \mid \exists \tilde{M} \in \mathcal{R}(\tilde{k}, n, m) : \|M - \tilde{M}\| \leq \texttt{rel\_eps}\|M\| \textit{ or } \|M - \tilde{M}\| \leq \texttt{abs\_eps}\}.$$

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->e[i];
  for(i=0; i<m; i++)
     for(j=0; j<c->kt; j++)
        c->b[i+j*m] = v->e[i+j*m] * s[j];
```

```
    free(s); free(u); free(v);
    del_rkmatrix(a_plus_b);
  }
```

5.2 Arithmetics in the $\mathcal{H}$-Matrix Format

5.2.1 Addition and Multiplication

In Definitions 5.7 and 5.8, 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 5.13 (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 5.14 *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.

Definition 5.15 (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 $\oplus$.

Implementation 5.16 (Formatted Addition) *Exercise 20.*

Definition 5.17 (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*

$$A \odot B := \mathcal{T}_k(A\cdot B).$$

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 5.18 (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 5.19 (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*

$$\mathcal{T}'_k : \mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, qk) \to \mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, k), \quad M \mapsto \tilde{M}$$

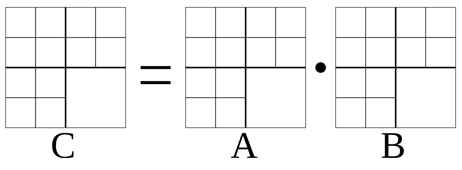
by $q-1$ times calling the truncation for a matrix with blockwise rank $2k$: let $M = \sum_{i=1}^{q} M_i$ be a decomposition of M into q matrices $M_i \in \mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, k)$. Then we define

$$\begin{aligned} \tilde{M}_1 &:= M_1, \\ \tilde{M}_i &:= \mathcal{T}_k(M_i + \tilde{M}_{i-1}), \quad i = 2, \ldots, q, \\ \tilde{M} &:= \tilde{M}_q. \end{aligned}$$

Example 5.20 (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 5.21 (Levels of the Tree) *Let T be a tree. We define the levels $\ell \in \mathbb{N}_0$ of T by*

$$T^{(\ell)} := \{t \in V(T) \mid \mathrm{level}(t) = \ell\}.$$

On each level $\ell \in \mathbb{N}_0$ we define

$$\mathcal{L}(T, \ell) := T^{(\ell)} \cap \mathcal{L}(T).$$

Lemma 5.22 *For any cluster tree $T_\mathcal{I}$ and $\ell \in \mathbb{N}$ there holds*

$$\mathcal{I} = \left(\dot{\bigcup_{v \in T_\mathcal{I}^{(\ell)}}} \hat{v} \right) \dot\cup \left(\dot{\bigcup_{v \in \mathcal{L}(T_\mathcal{I}, \ell-1)}} \hat{v} \right) \dot\cup \cdots \dot\cup \left(\dot{\bigcup_{v \in \mathcal{L}(T_\mathcal{I}, 0)}} \hat{v} \right).$$

Proof: Define $T'_\mathcal{I} := T_\mathcal{I} \setminus \cup_{j=\ell+1}^{\mathrm{depth}(T_\mathcal{I})} T_\mathcal{I}^{(j)}$ and apply Lemma 2.7. ∎

Definition 5.23 (Hierarchical Approximation) *Let $T_{\mathcal{I}\times\mathcal{I}}$ be a block cluster tree and $p := \mathrm{depth}(T_{\mathcal{I}\times\mathcal{I}})$. 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 5.1) by*

$$\begin{aligned}
M_p|_{\hat{t}\times\hat{s}} &:= \begin{cases} \mathcal{T}_k(M|_{\hat{t}\times\hat{s}}) & \textit{if } t\times s \textit{ inadmissible leaf,} \\ M|_{\hat{t}\times\hat{s}} & \textit{otherwise,} \end{cases} & , \\
M_{\ell-1}|_{\hat{t}\times\hat{s}} &:= \begin{cases} \mathcal{T}_k(M_\ell|_{\hat{t}\times\hat{s}}) & \textit{if } t\times s \in T_{\mathcal{I}\times\mathcal{I}}^{(\ell-1)}, \\ M_\ell|_{\hat{t}\times\hat{s}} & \textit{otherwise,} \end{cases} & \ell = p, \dots, 1, \\
M_\mathcal{H} &:= M_0.
\end{aligned}$$

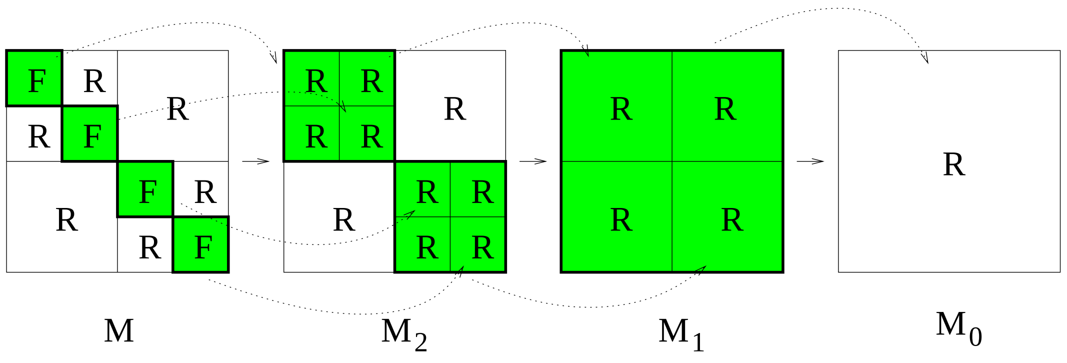


Figure 5.1: The matrix M is converted levelwise for each block: in the first step the `fullmatrix` blocks (F) to `rkmatrix` format (R), then the sons of a block to a single `rkmatrix`.

Lemma 5.24 (Hierarchical Approximation Error) *Let $T_{\mathcal{I}\times\mathcal{I}}$ be a block cluster tree, $p := \mathrm{depth}(T_{\mathcal{I}\times\mathcal{I}})$, $M \in \mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, k)$ and let $M_\mathcal{H}$ denote the hierarchical approximation of M. Then*

$$\|M - M_\mathcal{H}\|_F \le (2^{p+1} + 1)\|M - \mathcal{T}_k(M)\|_F.$$

Proof: We define the sets

$$\mathcal{R}(T_{\mathcal{I}\times\mathcal{I}}, \ell, k) := \{X \in \mathbb{R}^{\mathcal{I}\times\mathcal{I}} \mid \mathrm{rank}(X|_{\hat{t}\times\hat{s}}) \le k \quad \text{for all leaves } t\times s \in \mathcal{L}(T_{\mathcal{I}\times\mathcal{I}}) \text{ or } t\times s \in T_{\mathcal{I}\times\mathcal{I}}^{(\ell)}\}.$$

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 $\mathcal{R}(T_{\mathcal{I}\times\mathcal{I}}, \ell-1, k)$:

$$\forall X \in \mathcal{R}(T_{\mathcal{I}\times\mathcal{I}}, \ell, k): \quad \|M_\ell - M_{\ell-1}\|_F \le \|M_\ell - X\|_F. \tag{5.1}$$

In the first step (conversion of the `fullmatrix` blocks) this reads

$$\forall X \in \mathcal{R}(T_{\mathcal{I}\times\mathcal{I}}, p, k): \quad \|M - M_p\|_F \le \|M - X\|_F. \tag{5.2}$$

By induction we prove
$$\|M_\ell - \mathcal{T}_k(M)\|_F \le 2^{p-\ell}\|M_p - \mathcal{T}_k(M)\|_F$$
as follows. The start $\ell = p$ of the induction is trivial. The induction step $\ell \mapsto \ell - 1$ follows from
$$\|M_{\ell-1} - \mathcal{T}_k(M)\|_F \le \|M_{\ell-1} - M_\ell\|_F + \|M_\ell - \mathcal{T}_k(M)\|_F \overset{(5.1)}{\le} 2\|M_\ell - \mathcal{T}_k(M)\|_F.$$
Using this inequality, we can conclude that
$$\begin{aligned}
\|M - M_0\|_F &= \|M - \sum_{\ell=0}^{p-1}(M_\ell - M_{\ell+1}) \quad - M_p\|_F \\
&\le \|M - M_p\|_F + \sum_{\ell=0}^{p-1}\|M_\ell - M_{\ell+1}\|_F \\
&\overset{(5.1),(5.2)}{\le} \|M - \mathcal{T}_k(M)\|_F + \sum_{\ell=0}^{p-1}\|M_{\ell+1} - \mathcal{T}_k(M)\|_F \\
&\le \|M - \mathcal{T}_k(M)\|_F + \sum_{\ell=0}^{p-1} 2^{p-\ell-1}\|M_p - \mathcal{T}_k(M)\|_F \\
&\le 2^p\|M_p - \mathcal{T}_k(M)\|_F + \|M - \mathcal{T}_k(M)\|_F \\
&\le 2^p(\|M_p - M\|_F + \|M - \mathcal{T}_k(M)\|_F) + \|M - \mathcal{T}_k(M)\|_F \\
&\overset{(5.2)}{\le} (2^{p+1}+1)\|M - \mathcal{T}_k(M)\|_F.
\end{aligned}$$

■

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 5.20). 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, $\tilde{C} \ne C$ due to possible cancellation effects and the hierarchical approximation error.

Implementation 5.25 (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 */
  ...
  }
}
```

Now we are able to formulate the fast formatted multiplication.

Implementation 5.26 (Fast Multiplication of Hierarchical Matrices) *The matrices* `a`,`b`,`c` *are given in* `supermatrix` *format (underlying block cluster tree T_a based on T_1, T_2, T_b based on T_2, T_3 and T_c based on T_1, T_3). We compute* `c` $:=$ `c` $\oplus$ `a` $\odot$ `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 */
        ...
      }
    }
  }
```

5.2.2 Inversion

Definition 5.27 (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*

$$\widetilde{\text{Inv}} : \{M \in \mathbb{R}^{n\times n} \mid \text{rank}(M) = n\} \to \mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, k), \qquad M \mapsto \mathcal{T}_k(M^{-1}).$$

Since the (`fullmatrix`) inversion of M is rather expensive, we will give an algorithm that computes an approximation to $\mathcal{T}_k(M^{-1})$ without the need to invert M. This approximation will not necessarily be a best approximation.

The inversion is done by use of (4.4),

$$M^{-1} = \begin{bmatrix} M_{11}^{-1} + M_{11}^{-1}M_{12}S^{-1}M_{21}M_{11}^{-1} & -M_{11}^{-1}M_{12}S^{-1} \\ -S^{-1}M_{21}M_{11}^{-1} & S^{-1} \end{bmatrix},$$

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 $\text{Inv}_{\mathcal{H}}(M)$. This approximation is called the formatted inverse.

The generalisation to the case of `block_rows` $\times$ `block_cols` matrices is straightforward.

Implementation 5.28 (Formatted Inversion) *The procedure* `invert_supermatrix` *computes an approximate inverse* `si` *to the matrix* `s`*. The supermatrix* `swork` *is needed as workspace. All three matrices are assumed to be of the same structure (block cluster tree) and* `si`,`swork` *are initialised by* 0*. On exit the content of* `s` *and* `swork` *is overwritten.*

```
void
invert_supermatrix(psupermatrix si, psupermatrix s, psupermatrix swork){
  int bn = s->block_rows;
```

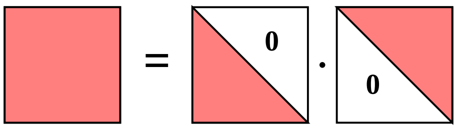
```
    int bm = s->block_cols;
    int i,j,l;
    psupermatrix *sw_e= swork->s, *s_e= s->s, *si_e= si->s;
    if(si->s==0x0 || s->s==0x0 || swork->s==0x0){
      /* inversion in the space of fullmatrices */
      invert_fullmatrix(si->f,s->f,swork->f);
    }else{
      /* lower triangular */
      for(l=0; l<bn; l++){
        /* invert diagonal */
        invert_supermatrix(si_e[l+bn*l],s_e[l+bn*l],sw_e[l+bn*l]);
        /* scale inverse with inverted diagonal */
        for(j=0; j<l; j++){
          mul_supermatrix(sw_e[l+bn*j],si_e[l+bn*l],si_e[l+bn*j]);
          copydata_supermatrix(sw_e[l+bn*j],si_e[l+bn*j]);
        }
        /* scale matrix with inverted diagonal */
        for(j=l+1; j<bm; j++){
          mul_supermatrix(sw_e[l+bn*j],si_e[l+bn*l],s_e[l+bn*j]);
          copydata_supermatrix(sw_e[l+bn*j],s_e[l+bn*j]);
        }
        /* eliminate lower part */
        for(i=l+1; i<bn; i++){
          /* subtract from inverse */
          for(j=0; j<=l; j++){
            mul_supermatrix(sw_e[i+bn*j],s_e[i+bn*l],si_e[l+bn*j]);
            scale_supermatrix(sw_e[i+bn*j],-1.0);
            addto_supermatrix(si_e[i+bn*j],sw_e[i+bn*j]);
          }
          /* subtract from matrix */
          for(j=l+1; j<bm; j++){
            mul_supermatrix(sw_e[i+bn*j],s_e[i+bn*l],s_e[l+bn*j]);
            scale_supermatrix(sw_e[i+bn*j],-1.0);
            addto_supermatrix(s_e[i+bn*j],sw_e[i+bn*j]);
          }
        }
      }
      /* upper triangular */
      for(l=bn-1; l>=0; l--){
        for(i=l-1; i>=0; i--){
          for(j=0; j<bm; j++){
            mul_supermatrix(sw_e[i+bn*j],s_e[i+bn*l],si_e[l+bn*j]);
            scale_supermatrix(sw_e[i+bn*j],-1.0);
            addto_supermatrix(si_e[i+bn*j],sw_e[i+bn*j]);
          }
        }
      }
    }
  }
```

5.2.3 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

$$A \approx LU$$

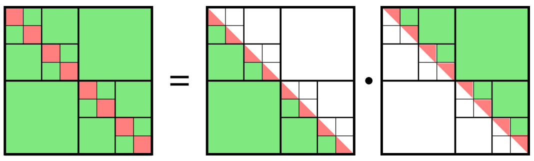


of the matrix A. Depending on the application one could want a decomposition such that $\|A - LU\|$ is of the size of the discretisation error or just a coarse approximation in order to precondition the linear system and use a simple iterative solver, e.g., GMRES:

$$(LU)^{-1}Ax = (LU)^{-1}b$$

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.

5.3 Exercises

5.3.1 Theory

Exercise 17 (Truncation to the Hierarchical Matrix Format) *Prove the statement of Lemma 5.14, i.e.,*

$$\|M - \tilde{M}\|_F = \min_{M' \in \mathcal{H}(T_{\mathcal{I}\times\mathcal{I}},k)} \|M - M'\|_F.$$

Exercise 18 (Hierarchical Approximation) *Let $T_{\mathcal{I}\times\mathcal{I}}$ be a block cluster tree, $p := \mathrm{depth}(T_{\mathcal{I}\times\mathcal{I}})$, $M \in \mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, k)$ and let $M_{\mathcal{H}}$ denote the hierarchical approximation of M (Definition 5.23). Prove that*

$$\|M - M_{\mathcal{H}}\|_2 \leq (2^{\frac{3p}{2}+1} + 2^{p/2})\|M - \mathcal{T}_k(M)\|_2.$$

Hint: Don't be confused, this is an easy exercise.

Exercise 19 (Cholesky Decomposition) *Derive an algorithm for the $\mathcal{H}$-LDL^T decomposition of a symmetric matrix A with lower triangular matrix L and diagonal matrix D.*

5.3.2 Practice

Exercise 20 (Addition of `supermatrix` structures) *Let $A, B \in \mathcal{H}(T_{\mathcal{I}\times\mathcal{I}}, k)$ be given in $\mathcal{H}$-matrix representation implemented by use of the* `supermatrix` *structure. Implement the formatted addition by a recursive procedure.*

```
void
add_supermatrix(psupermatrix c, psupermatrix a, psupermatrix b);
```

Exploit the fact that the three matrices `a,b,c` *are assumed to be of the same $\mathcal{H}$-matrix format. Test your implementation with the program* `example_add` *by reading the two matrices* `example_add_matrix1.hma` *and* `example_add_matrix2.hma` *from a file with the procedure* `read_supermatrix` *(*`supermatrix.h`*) and compare the result with the matrix* `example_add_matrix1+2.hma`*. For the comparison the function* `norm2diff_supermatrix` *(*`supermatrix.h`*) might be helpful.*

Exercise 21 (Preconditioned Conjugate Gradient Method) *The solver that we used in* `example_1d.c` *from Exercise 6 to compute the solution x to $Ax - b = 0$, $A \in \mathbb{R}^{n\times n}$, is the conjugate gradient method:*
Start:

$$x^0 := 0, \quad r^0 := b, \quad p^0 := r^0.$$

Iteration $i = 1, \dots, n-1$:

$$\begin{aligned} \lambda &:= \langle r^{i-1}, p^{i-1}\rangle / \langle Ap^{i-1}, p^{i-1}\rangle, \\ r^i &:= r^{i-1} - \lambda A p^{i-1}, \\ p^i &:= r^i - p^{i-1}\langle r^i, Ap^{i-1}\rangle / \langle Ap^{i-1}, p^{i-1}\rangle, \\ x^i &:= x^{i-1} + \lambda p^i. \end{aligned}$$

Stop if $\|r^i\| < \epsilon$, where ϵ is the desired accuracy, i.e., $\|Ax - b\| < \epsilon$.

The number of iterations necessary to reach the accuracy ϵ increases if the condition number of A increases. A method to circumvent this is to **precondition** *A, i.e., to solve the system*

$$PAx = Pb$$

for a matrix P such that $\mathrm{cond}(PA) \approx 1$.

Extend the procedure

```
void
solve_cg_supermatrix(psupermatrix s, double *rhs, double *u,
                     double eps, double maxsteps,
                     psupermatrix prec, int prec_type,
                     int verbosity);
```

that performs the conjugate gradient method up to accuracy $\epsilon := 10^{-13}$ (or $i = maxsteps$) and preconditions the matrix `s` *by* `prec`*. The right-hand side b is denoted by* `rhs`*, the unknown solution is stored in the already allocated array* `u`*. The variable* `prec_type` *is used to distinguish between different types of preconditioners (see next exercise), for the beginning* `prec` *is just a* `supermatrix` *and* `prec_type=1`*. The variable* `verbosity` *tells the procedure wether or not it should print information during the iteration.*

Hint*: Do not compute the matrix-matrix product* `prec`$\odot$`s` *and allow the preconditioner to be* `prec=0x0`*, i.e., unpreconditioned.*

Exercise 22 (Numerical Test) *A preconditioner P can be obtained in different ways.*

1. *Use the formatted inversion for hierarchical matrices:*

   ```
   void
   invert_supermatrix(psupermatrix si, psupermatrix sc, psupermatrix swork);
   ```

   The inversion need not be that accurate, therefore the stiffness matrix `s` *is replaced by a coarser assembled matrix* `sc` *(the rank k is smaller, e.g. $k_coarse = 3$).*

   Use the approximate inverse `si` *as preconditioner for the conjugate gradient method and compute the solution to the model problem in* `example_1d.c`*.*

2. *Modify the procedure* `solvepreconditionedcg_supermatrix` *such that the matrix* `winv` *is given in Cholesky factorisation and use the subroutine* `solve_cholesky` *in order to evaluate the matrix* `winv`$=(LL^T)^{-1}$. *Test the procedure for the model problem from* `example_1d.c`.

   Hint*: The stiffness matrix for the integral operator from Section 1.1 is not positive.*

Chapter 6

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 [14] (german) and [16] (english). For the one-dimensional case the complexity estimates can be simplified as in [23], and for a two- and three-dimensional model problem the storage, matrix-vector mutliplication and addition can be estimated as in [24].

6.1 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

$$N_{St,R}(k,n,m) = k(n+m) \tag{6.1}$$

while in the `fullmatrix` representation we have

$$N_{St,F}(k,n,m) = nm. \tag{6.2}$$

6.1.1 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 6.1 (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 6.2 (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 $\mathcal{O}(\min(n,m)\max(n,m)k)$.*

6.1.2 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:

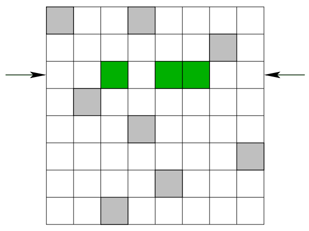
$$\begin{aligned} RM &= AB^TM = \sum_{\nu=1}^k A_\nu (M^T B_\nu)^T, \\ MR &= MAB^T = \sum_{\nu=1}^k (MA_\nu)B_\nu^T. \end{aligned}$$

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 $\mathcal{O}((n+m)k^2)$.

6.2 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,

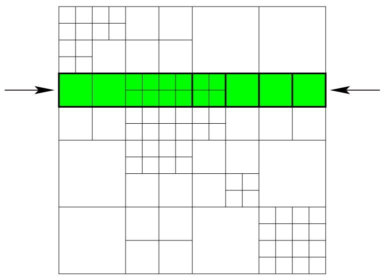


$$c := \max_{i\in\mathcal{I}} \#\{j \in \mathcal{I} \mid M_{ij} \neq 0\},$$

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 6.3, 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 6.3 (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*



$$C_{\mathrm{sp}} := \max\Big\{ \max_{r\in T_{\mathcal{I}}} \#\{s \in T_{\mathcal{I}} \mid r\times s \in T_{\mathcal{I}\times\mathcal{I}}\}, \max_{s\in T_{\mathcal{I}}} \#\{r \in T_{\mathcal{I}} \mid r\times s \in T_{\mathcal{I}\times\mathcal{I}}\} \Big\}.$$

Definition 6.4 (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 6.5 (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*

$$N_{\mathrm{St}}(T,k) \le 2C_{\mathrm{sp}}(p+1)\max\{k,n_{\min}\}\#\mathcal{I}.$$

Proof:

$$\begin{aligned}
N_{\mathrm{St}}(T,k) &\overset{(6.1),(6.2)}{=} \sum_{t\times s\in\mathcal{L}^+(T)} k(\#\hat{t}+\#\hat{s}) + \sum_{t\times s\in\mathcal{L}^-(T)} \#\hat{t}\cdot\#\hat{s}\\
&\le \sum_{t\times s\in\mathcal{L}^+(T)} k(\#\hat{t}+\#\hat{s}) + \sum_{t\times s\in\mathcal{L}^-(T)} n_{\min}(\#\hat{t}+\#\hat{s})\\
&\le \sum_{i=0}^{p}\sum_{t\times s\in T^{(i)}} \max\{k,n_{\min}\}(\#\hat{t}+\#\hat{s})\\
&= \sum_{i=0}^{p}\sum_{t\times s\in T^{(i)}} \max\{k,n_{\min}\}\#\hat{t} + \sum_{i=0}^{p}\sum_{t\times s\in T^{(i)}} \max\{k,n_{\min}\}\#\hat{s}\\
&\le 2C_{\mathrm{sp}}\max\{k,n_{\min}\}\sum_{i=0}^{p}\sum_{t\in T^{(i)}}\#\hat{t}\\
&\overset{\text{Lemma 5.22}}{\le} 2C_{\mathrm{sp}}\max\{k,n_{\min}\}\sum_{i=0}^{p}\#\mathcal{I}\\
&= 2C_{\mathrm{sp}}\max\{k,n_{\min}\}(p+1)\#\mathcal{I}.
\end{aligned}$$

■

Lemma 6.6 (Matrix-Vector Multiplication) *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 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*

$$N_{\mathcal{H}\cdot v}(T,k) \le 2N_{\mathrm{St}}(T,k).$$

Proof: Consider the matrix-vector product blockwise and use (6.1), (6.2) together with the respective counterparts for the matrix-vector product. ■

6.2.1 Truncation

Lemma 6.7 (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*
$$\#T_{\mathcal{I}} \;\leq\; 2\#\mathcal{I}, \qquad \#T \;\leq\; 2C_{\mathrm{sp}}\#\mathcal{I}. \tag{6.3}$$
2. *If $\#\operatorname{sons}(t) \neq 1$ is not necessarily fulfilled then*
$$\#T_{\mathcal{I}} \;\leq\; 2p\#\mathcal{I}, \qquad \#T \;\leq\; 2pC_{\mathrm{sp}}\#\mathcal{I}. \tag{6.4}$$

Lemma 6.8 (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:
$$\begin{aligned} N_{\mathcal{T}}(T,k) &= \sum_{t\times s\in\mathcal{L}^+(T)} N_{\mathcal{T}}(k,\#\hat{t},\#\hat{s}) \overset{\text{Lemma 6.1}}{\leq} \sum_{t\times s\in\mathcal{L}^+(T)} 6k(\#\hat{t}+\#\hat{s}) + 23k^3\#\mathcal{L}(T) \\ &\leq 6kN_{\mathrm{St}}(T,k) + 23k^3\#\mathcal{L}(T). \end{aligned}$$

■

Lemma 6.9 (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 6.8 for rank $2k$. ■

Lemma 6.10 (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 $C_{\mathrm{sons}} \geq 2$ sons (typically $C_{\mathrm{sons}} = 2$) and $n_{min} \leq k$ (for simplification). Then the complexity to compute the hierarchical approximation (cf. Definition 5.23) is bounded by*
$$N_{\mathcal{H}apx}(T,k) \;\leq\; 12C_{\mathrm{sp}}C_{\mathrm{sons}}^2(p+1)k^2\#\mathcal{I} \;+\; 23C_{\mathrm{sons}}^3k^3\#T.$$

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 $\#\hat{t} \leq n_{\mathrm{min}}$ or $\#\hat{s} \leq n_{\mathrm{min}}$ 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 an $\hat{t}\times\hat{s}$ `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_{sons}^2k. The truncation to rank k is according to Lemma 6.1 of complexity $6C_{\mathrm{sons}}^2k^2(\#\hat{s}+\#\hat{t}) + 23C_{\mathrm{sons}}^3k^3$. For all nodes this sums up to
$$\begin{aligned} N_{\mathcal{H}apx}(T,k) &\leq \sum_{t\times s\in\mathcal{L}^-(T)} 21n_{min}^2(\#\hat{s}+\#\hat{t}) + \sum_{t\times s\in T\setminus\mathcal{L}^-(T)} \left(6C_{\mathrm{sons}}^2k^2(\#\hat{s}+\#\hat{t}) + 23C_{\mathrm{sons}}^3k^3\right) \\ &\leq \sum_{t\times s\in T} \left(6C_{\mathrm{sons}}^2k^2(\#\hat{s}+\#\hat{t}) + 23C_{\mathrm{sons}}^3k^3\right) \\ &\leq 2C_{\mathrm{sp}}(p+1)\left(6C_{\mathrm{sons}}^2k^2\#\mathcal{I}\right) + 23C_{\mathrm{sons}}^3k^3\#T. \end{aligned}$$

■

6.2.2 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.

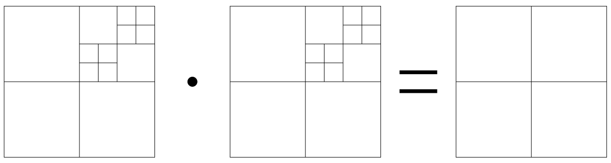
Lemma 6.11 (Addition) *Let $T = T_{\mathcal{I}\times\mathcal{I}}$ be a block cluster tree of depth p with sparsity constant C_{sp}. Then the complexity of the formatted addition of two matrices from $\mathcal{H}(T,k)$ is bounded by*

$$N_{\oplus}(T,k) \le 24kN_{\text{St}}(T,k) + 184k^3\#\mathcal{L}(T).$$

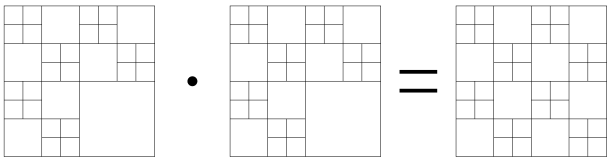
6.2.3 Multiplication

The multiplication is a much more complicated operation than the addition, as we have already seen in Examples 5.18, 5.20. In order to illustrate what might happen during the multiplication we will take a look at three typical examples.

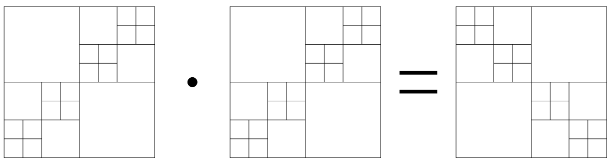
It may happen that the structure of the product of two matrices from $\mathcal{H}(T_{\mathcal{I}\times\mathcal{I}},k)$ is "coarser" than $T_{\mathcal{I}\times\mathcal{I}}$:



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:

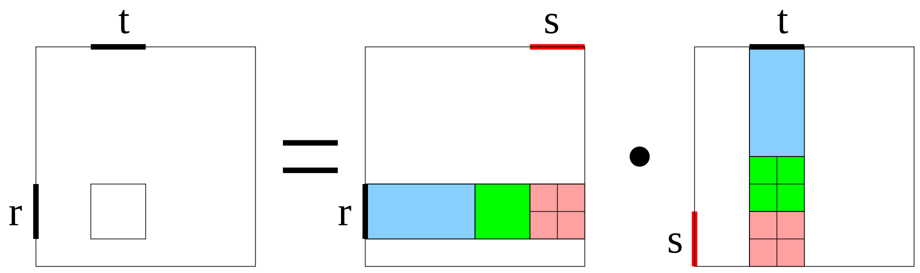


This poses the question how one can efficiently describe the structure of the product. The product tree $T\cdot T$ should be a block cluster tree based on the cluster tree $T_{\mathcal{I}}$, i.e.,

$$\text{root}(T\cdot T) := \mathcal{I}\times\mathcal{I}.$$

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 not a leaf in T, then the node $r \times t$ in the product tree must not be a leaf. This leads to the definition

Definition 6.12 (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 $\mathrm{root}(T \cdot T) := \mathcal{I} \times \mathcal{I}$ and for each node $r \times t$ of the product tree the successors*

$$\mathrm{sons}(r \times t) := \left\{ r' \times t' \;\middle|\; \exists s, s' \in T_{\mathcal{I}} : \; r' \times s' \in \mathrm{sons}_T(r \times s), s' \times t' \in \mathrm{sons}_T(s \times t) \right\}.$$

The sparsity of the block cluster tree T carries over to the product tree $T \cdot T$.

Lemma 6.13 *The product tree $T \cdot T$ is a block cluster tree of depth at most $\mathrm{depth}(T)$. 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:*

$$C_{\mathrm{sp}}(T \cdot T) \le C_{\mathrm{sp}}(T)^2.$$

Proof: Due to the symmetry of the sparsity we only give a rowwise bound. Let $r \in T_{\mathcal{I}}$. Then

$$\begin{aligned} \{t \in T_{\mathcal{I}} \mid r \times t \in T \cdot T\} &\subset \{t \in T_{\mathcal{I}} \mid \exists s \in T_{\mathcal{I}} : r \times s \in T, s \times t \in T\}, \\ \#\{t \in T_{\mathcal{I}} \mid r \times t \in T \cdot T\} &\le \sum_{s \in T_{\mathcal{I}}, r \times s \in T} \#\{t \in T_{\mathcal{I}} \mid s \times t \in T\} \\ &\le C_{\mathrm{sp}}(T) C_{\mathrm{sp}}(T). \end{aligned}$$

■

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 6.14 (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 $j \le \ell$ 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 6.15 (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*

$$\mathcal{U}(r \times t, j) \;:=\; \left\{ s \in T_{\mathcal{I}} \mid \mathcal{F}^j(r) \times s \in T, s \times \mathcal{F}^j(t) \in T \textit{ and at least one of the two is a leaf} \right\}.$$

Then for two matrices $A, B \in \mathcal{H}(T, k)$ and each $r \times t \in T \cdot T$ there holds

$$\mathcal{I} = \dot{\bigcup_{j=0,\ldots,\ell}} \; \dot{\bigcup_{s \in \mathcal{U}(r \times t, j)}} \hat{s} \tag{6.5}$$

$$(AB)|_{\hat{r} \times \hat{t}} = \sum_{j=0}^{\ell} \sum_{s \in \mathcal{U}(r \times t, j)} A|_{\hat{r} \times \hat{s}} B|_{\hat{s} \times \hat{t}}. \tag{6.6}$$

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 5.22 yields the disjointness.

(**Disjointness w.r.t.** j) Let $s_1 \in \mathcal{U}(r \times t, j_1)$ and $s_2 \in \mathcal{U}(r \times t, j_2)$, $j_2 \le j_1$ 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

$$\widehat{\mathcal{F}^{j_1}(r)} \times \hat{s}_1 \subset \widehat{\mathcal{F}^{j_2}(r)} \times \hat{s}_2, \qquad \hat{s}_1 \times \widehat{\mathcal{F}^{j_1}(t)} \subset \hat{s}_2 \times \widehat{\mathcal{F}^{j_2}(t)}. \tag{6.7}$$

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 (6.7) 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 $s \in \text{sons}(\mathcal{F}^0(r))$ 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 6.16 (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*

$$\tilde{k} \le (p+1) C_{\text{sp}} \max\{k, n_{\min}\}. \tag{6.8}$$

The exact multiplication can be performed with complexity

$$N_{\mathcal{H},\cdot}(T,k) \le 4(p+1) C_{\text{sp}}^2 \max\{k, n_{\min}\} N_{\text{St}}(T,k).$$

Proof: (Rank) Let $A, B \in \mathcal{H}(T,k)$ and $r \times t \in \mathcal{L}(T \cdot T, \ell)$. Due to (6.6), 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 $\tilde{k} \le (p+1)\max_{j=0}^{\ell} \#\mathcal{U}(r \times t, j) \max\{k, n_{\min}\}$. The cardinality of $\mathcal{U}(r \times t, j)$ is bounded by

$$\#\mathcal{U}(r \times t, j) \quad \le \quad \#\{s \in T_\mathcal{I} \mid \mathcal{F}^j(r) \times s \in T\} \le C_{\text{sp}}(T)$$

which yields $\#\mathcal{U}(r \times s, j) \le C_{\text{sp}}(T)$.

b) (Complexity) Using the representation formula (6.6), we have to compute the products $A|_{\hat{r}\times\hat{s}} B|_{\hat{s}\times\hat{t}}$ that consist (due to the definition of $\mathcal{U}(r \times t, j)$) of $\max\{k, n_{\min}\}$ matrix-vector products. In the following, the expressions $N_{\mathcal{H},St}(T_{\hat{r}\times\mathcal{I}}, k)$ and $N_{\mathcal{H},St}(T_{\mathcal{I}\times\hat{t}}, k)$ appear which denote the storage requirements for a submatrix to the index set $\hat{r} \times J$ and $J \times \hat{t}$ of a matrix in $\mathcal{H}(T,k)$. We use the abbreviation $\kappa := \max\{k, n_{\min}\}$ and conclude that

$$\begin{aligned}
N_{\mathcal{H},\cdot}(T,k) &\overset{\text{Lem.6.6}}{\le} \sum_{r\times t \in \mathcal{L}(T\cdot T)} \sum_{j=0}^{p} \sum_{s \in \mathcal{U}(r\times t, j)} \kappa \max\{2N_{\text{St}}(T_{r\times s}, k), 2N_{\text{St}}(T_{s\times t}, k)\} \\
&\overset{(6.6)}{\le} \sum_{r\times t \in \mathcal{L}(T\cdot T)} 2\kappa \max\{N_{\text{St}}(T_{\hat{r}\times\mathcal{I}}, k), N_{\text{St}}(T_{\mathcal{I}\times\hat{t}}, k)\} \\
&\le 2\kappa \sum_{j=0}^{p} \left(\sum_{r\times t \in \mathcal{L}(T\cdot T, j)} N_{\text{St}}(T_{\hat{r}\times\mathcal{I}}, k) + \sum_{r\times t \in \mathcal{L}(T\cdot T, j)} N_{\text{St}}(T_{\mathcal{I}\times\hat{t}}, k) \right) \\
&\overset{\text{Lem.6.13}}{\le} 2\kappa \sum_{j=0}^{p} \left(C_{\text{sp}}^2 \sum_{r \in T_\mathcal{I}^{(j)}} N_{\text{St}}(T_{\hat{r}\times\mathcal{I}}, k) + C_{\text{sp}}^2 \sum_{t \in T_\mathcal{I}^{(j)}} N_{\text{St}}(T_{\mathcal{I}\times\hat{t}}, k) \right) \\
&\le 4\kappa(p+1) C_{\text{sp}}^2 N_{\text{St}}(T,k),
\end{aligned}$$

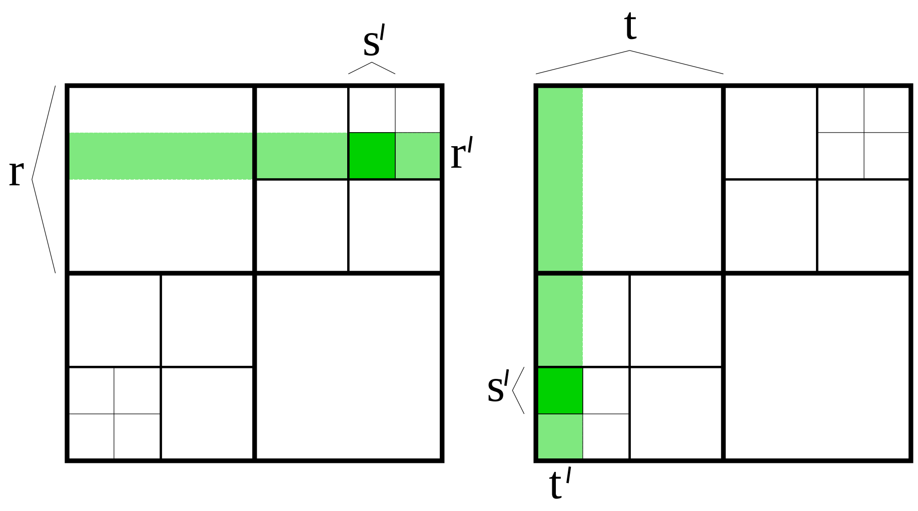


Figure 6.1: The idempotency constant $C_{\text{sp}}(r \times t)$ of the leaf $r \times t$ is 9.

proving the last estimate. ∎

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 6.17 (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 6.1) $C_{\text{id}}(r \times t)$ and* idempotency constant *$C_{\text{id}}(T)$ by*

$$\begin{aligned} C_{\text{id}}(r \times t) &:= \#\{r' \times t' \mid r' \in S^*(r), t' \in S^*(t) \text{ and } \exists s' \in T_{\mathcal{I}} : r' \times s' \in T, s' \times t' \in T\}, \\ C_{\text{id}}(T) &:= \max_{r \times t \in \mathcal{L}(T)} C_{\text{id}}(r \times t). \end{aligned}$$

If the tree T is fixed, the short notation C_{id} is used instead of $C_{\text{id}}(T)$.

Theorem 6.18 (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*

$$\tilde{k} \leq C_{\text{id}} C_{\text{sp}} (p+1) k.$$

The formatted multiplication $\odot^{\text{best}} : \mathcal{H}(T, k) \times \mathcal{H}(T, k) \to \mathcal{H}(T, k')$ for any $k' \in \mathbb{N}_0$ is defined as the exact multiplication followed by the truncation $\mathcal{T}_{k'}$ of Lemma 5.13 and can be computed with complexity

$$N_{\odot,\text{best}}(T, k) \leq 43 C_{\text{id}}^3 C_{\text{sp}}^3 k^3 (p+1)^3 \max\{\#\mathcal{I}, \#\mathcal{L}(T)\}$$

by truncating the exact product. Using the fast truncation $\mathcal{T}'_{k'}$ of Defintion 5.19, the complexity can be reduced to

$$N_{\odot}(T, k) \leq 56 C_{\text{sp}}^2 \max\{C_{\text{id}}, C_{\text{sp}}\} k^2 (p+1)^2 \#\mathcal{I} + 184 C_{\text{sp}} C_{\text{id}} k^3 (p+1) \#\mathcal{L}(T).$$

We call this mapping $\odot$ or $\odot^{\text{fast}}$ in contrast to $\odot^{\text{best}}$ from above.

Proof: (a. Rank) Due to (6.8), 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 $N_{+}, N_{+}^{\mathrm{fast}}$ for the (fast) truncation of the `rkmatrix` blocks to leaves $\mathcal{L}^{+}(T)$ with rank k'.

(b1. N_{mul}**)** According to Theorem 6.16 and Lemma 6.5, the exact product using the `rkmatrix` representation with rank $\tilde{k}$ each leaf can be computed with complexity $4C_{\mathrm{sp}}^3(p+1)^2k^2\#\mathcal{I}$.

(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 $2\tilde{k}\#\hat{r}\#\hat{t}$:

$$\begin{aligned}
N_{-} &\leq \sum_{r\times t\in\mathcal{L}^{-}(T)} 2\tilde{k}\#\hat{t}\#\hat{s}\\
&\leq \sum_{r\times t\in\mathcal{L}(T)} 2\tilde{k}n_{\min}(\#\hat{r}+\#\hat{t})\\
&\leq \sum_{i=0}^{p}\sum_{r\times t\in\mathcal{L}(T,i)} 2\tilde{k}n_{\min}(\#\hat{r}+\#\hat{t})\\
&\overset{\text{Lemma 5.22}}{\leq} 4(p+1)C_{\mathrm{sp}}\tilde{k}n_{\min}\#\mathcal{I}\\
&\leq 4(p+1)^2C_{\mathrm{sp}}^2C_{\mathrm{id}}kn_{\min}\#\mathcal{I}.
\end{aligned}$$

(b3. N_{+}**)** For each leaf in $\mathcal{L}^{+}(T)$ we truncate the `rkmatrix` block of rank $\tilde{k}$ to rank k using Lemma 6.8 for the truncation or Lemma 6.9 for the fast truncation:

$$\begin{aligned}
N_{+} &\overset{\text{Lem.6.8}}{\leq} 6\tilde{k}N_{\mathrm{St}}(T,\tilde{k}) + 23(\tilde{k})^3\#\mathcal{L}(T)\\
&\overset{\text{Lem.6.5}}{\leq} 12C_{\mathrm{sp}}^3C_{\mathrm{id}}^2k^2(p+1)^3\#\mathcal{I} + 23C_{\mathrm{sp}}^3C_{\mathrm{id}}^3k^3(p+1)^3\#\mathcal{L}(T)\\
&\leq 35C_{\mathrm{sp}}^3C_{\mathrm{id}}^3k^3(p+1)^3\max\{\#\mathcal{I},\#\mathcal{L}(T)\},
\end{aligned}$$

$$\begin{aligned}
N_{+}^{\mathrm{fast}} &\overset{\text{Lem.6.9}}{\leq} C_{\mathrm{sp}}C_{\mathrm{id}}(p+1)\left(24kN_{\mathrm{St}}(T,k) + 184k^3\#\mathcal{L}(T)\right)\\
&\overset{\text{Lem.6.5}}{\leq} 48C_{\mathrm{sp}}^2C_{\mathrm{id}}k^2(p+1)^2\#\mathcal{I} + 184C_{\mathrm{sp}}C_{\mathrm{id}}k^3(p+1)\#\mathcal{L}(T).
\end{aligned}$$

■

6.2.4 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 6.19 (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 (Implementation 5.28) 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 $r_1, \ldots, r_n$ denote the sons of the root $\mathcal{I}$ of $T_{\mathcal{I}}$. For the inversion of the matrix we call the multiplication `mul_supermatrix` for all combinations of blocks (r_i, r_ℓ, r_j), $i, \ell, j \in \{1, \dots, n\}$, where the combination $i = \ell = j$ stands for the inversion which is by induction at most of the same complexity as the multiplication. This is exactly what is done for the computation of the product of two matrices in `supermatrix` format (see Implementation 5.26). Additionally, we have to call the formatted addition `add_supermatrix` in the blocks $r_i \times r_j$, again the same for the product. ■

6.3 Sparsity and Idempotency of the Block Cluster Tree $T_{\mathcal{I}\times\mathcal{I}}$

6.3.1 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. We denote the Chebyshev centre of such a support Ω_i by m_i. In practice the vertex m_i can be any point inside the support of 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 6.20 (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 $\mathrm{root}(T_{\mathcal{I}})$ 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 Chebyshev centre of Ω_i is contained in the respective box, a box B_t containing Ω_t has to be $\frac{1}{2}\mathrm{diam}(\Omega_i)$ larger than B_t. We define the local meshwidth h_t and box B_t by

$$h_t := \max_{i \in \hat{t}} \mathrm{diam}(\Omega_i), \qquad B_t := C_t + \frac{1}{2}[-h_t, h_t]^2. \tag{6.9}$$

Lemma 6.21 *For any two nodes $t, s \in T_{\mathcal{I}}^{(\ell)}$ there holds*

$$\begin{aligned}
\Omega_t &\subset B_t\\
\mathrm{diam}(B_t) &= \sqrt{2}(2^{-\ell} + h_t)H &(6.10)\\
\mathrm{dist}(B_t, B_s) &\geq \mathrm{dist}(C_t, C_s) - \frac{1}{2}\sqrt{2}(h_t + h_s). &(6.11)
\end{aligned}$$

Proof: Let $i \in t$. By Construction 6.20 we get $m_i \in C_t$. The elements in the support Ω_i have a distance of at most $\frac{1}{2}\mathrm{diam}(\Omega_i)$ from the centre m_i and thus $\Omega_i \subset B_t$. The second and third part follow from the definition of B_t. ■

6.3.2 Construction of the Block Cluster Tree $T_{\mathcal{I}\times\mathcal{I}}$

Based on the cluster tree $T_\mathcal{I}$ from Construction 6.20 and the admissibility condition (2.3) we define the block cluster tree $T := T_{\mathcal{I}\times\mathcal{I}}$ as follows.

A product index set $r \times s$ with corresponding boxes C_r and C_s is called admissible, if

$$\min\{\widetilde{\mathrm{diam}}(r), \widetilde{\mathrm{diam}}(s)\} \le \widetilde{\mathrm{dist}}(r,s), \tag{6.12}$$

where the modified distance and diameter are

$$\begin{aligned}\widetilde{\mathrm{diam}}(t) &:= \mathrm{diam}(B_t),\\ \widetilde{\mathrm{dist}}(r,s) &:= \mathrm{dist}(B_r, B_s).\end{aligned}$$

If a product $r \times s$ is admissible with respect to (6.12) then the corresponding domain $\Omega_r \times \Omega_s$ is admissible with respect to the standard admissibility condition (2.3).

This modified admissibility condition is used to construct the block cluster tree $T_{\mathcal{I}\times\mathcal{I}}$ as usual.

Construction 6.22 (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 $\widehat{\mathrm{root}(T)} := \mathcal{I}\times\mathcal{I}$ and for each vertex $r \times s \in T$ the set of successors*

$$\mathrm{sons}(r \times s) := \begin{cases} \{r' \times s' \mid r' \in \mathrm{sons}(r), s' \in \mathrm{sons}(s)\} & \textit{if } \#r > n_{\min} \textit{ and } \#s > n_{\min} \textit{ and } r\times s \textit{ is inadmissible,}\\ \emptyset & \textit{otherwise.}\end{cases}$$

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 6.23 *Let $T := T_{\mathcal{I}\times\mathcal{I}}$ be the block cluster tree of depth $p \ge 1$ built from the cluster tree $T_\mathcal{I}$ by Construction 6.22. Then the sparsity constant (cf. Definition 6.3) C_{sp} of T is bounded by*

$$C_{\mathrm{sp}} \le 4 \max_{r\in T_\mathcal{I}} \#\{s \in T_\mathcal{I} \mid r\times s \in T\setminus\mathcal{L}(T) \textit{ and } r \times s \textit{ is inadmissible}\}.$$

Proof: Let $r \times s \in T^{(\ell)}$. Then $r \times s$ is either the root of T or the father element $\mathcal{F}(r)\times\mathcal{F}(s)$ is inadmissible due to Construction 6.22. ■

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 6.24 (Locality) *We assume that the supports are locally separated in the sense that there exist two constants C_{sep} and $n_{\min}$ such that*

$$\max_{i\in\mathcal{I}} \#\{j \in \mathcal{I} \mid \mathrm{dist}(\Omega_i,\Omega_j) \le C_{\mathrm{sep}}^{-1}\mathrm{diam}(\Omega_i)\} \le n_{\min}. \tag{6.13}$$

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 6.25 (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 6.20 and assume that (6.13) holds for some constants $C_{\mathrm{sep}}, n_{\min}$. Let $T := T_{\mathcal{I}\times\mathcal{I}}$ be the block cluster tree from Construction 6.20 and 6.22. Then the following statements hold:*
(a) All leaves $t \times s \in \mathcal{L}(T)$ are either admissible with respect to (6.12) or $\min\{\#\hat{t}, \#\hat{s}\} \le n_{\min}$.
(b) The depth of the tree is bounded by

$$\operatorname{depth}(T) \le 1 + \log_2\left((1 + C_{\mathrm{sep}})\sqrt{2}H/h_{\min}\right).$$

(c) The sparsity constant is bounded by

$$C_{\mathrm{sp}} \le (4 + 16C_{\mathrm{sep}} + 8\sqrt{2}\eta^{-1}(1 + C_{\mathrm{sep}}))^2.$$

(d) The idempotency constant is bounded by

$$C_{\mathrm{id}} \le ((4 + 4\eta)(1 + C_{\mathrm{sep}}))^4.$$

Proof: (a) Holds by Construction 6.22.

(b) Let $t \in T_{\mathcal{I}}^{(\ell)}$ be a non-leaf node. Then $\#\hat{t} > n_{\min}$ such that

$$\operatorname{diam}(B_t) \overset{(6.10)}{=} \sqrt{2}(2^{-\ell} + h_t)H \overset{(6.13)}{\le} \sqrt{2}(2^{-\ell} + C_{\mathrm{sep}}\operatorname{diam}(C_t))H = \sqrt{2}(1 + C_{\mathrm{sep}})2^{-\ell}H \tag{6.14}$$

while $\operatorname{diam}(B_t) \ge \operatorname{diam}(\Omega_i) \ge h_{\min}$. This yields $2^\ell \le \sqrt{2}(1 + C_{\mathrm{sep}})H/h_{\min}$.

(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

$$\begin{aligned}
\operatorname{diam}(B_t) &\overset{(6.14)}{\le} \sqrt{2}(1 + C_{\mathrm{sep}})2^{-\ell}H,\\
\operatorname{dist}(B_t, B_s) &\overset{(6.11)}{\ge} \operatorname{dist}(C_t, C_s) - \frac{1}{2}\sqrt{2}(h_t + h_s)\\
&> j2^{-\ell}H - \frac{1}{2}\sqrt{2}C_{\mathrm{sep}}(\operatorname{diam}(C_t) + \operatorname{diam}(C_s))\\
&= j2^{-\ell}H - C_{\mathrm{sep}}2^{1-\ell}H.
\end{aligned}$$

If $t \times s$ is not admissible, then (6.12) enables us to further estimate

$$\sqrt{2}(1 + C_{\mathrm{sep}})2^{-\ell}H > \eta(j2^{-\ell}H - 2C_{\mathrm{sep}}2^{-\ell}H)$$

which yields

$$j < \eta^{-1}\left(\sqrt{2}(1 + C_{\mathrm{sep}}) + 2\eta C_{\mathrm{sep}}\right) =: j_{\max}.$$

As a consequence the number of nodes $s \in T_{\mathcal{I}}^{(\ell)}$ not admissible to t is bounded by $(1 + 2j_{\max})^2$. The number of successors of s is bounded by 4 such that on level $\ell + 1$ there are at most $4(1 + 2j_{\max})^2$ nodes inadmissible to a son of t. Lemma 6.23 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 $q := \lceil(\log_2(2(1 + \eta)(1 + C_{\mathrm{sep}}))\rceil$. 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

$$\widetilde{\operatorname{diam}}(u) \overset{(6.14)}{\le} \sqrt{2}(1 + C_{\mathrm{sep}})2^{-q-\ell}H \overset{\mathrm{Def.}q}{\le} \frac{1}{2}\sqrt{2}(1 + \eta)^{-1}2^{-\ell}H. \tag{6.15}$$

Then we can estimate

$$\begin{aligned}
\widetilde{\operatorname{diam}}(s') &\overset{(6.15)}{\leq} \frac{1}{2}\sqrt{2}(1-\eta(\eta+1)^{-1})2^{-\ell}H \\
&= \frac{1}{2}\sqrt{2}2^{-\ell}H - \eta\frac{1}{2}\sqrt{2}2^{-\ell}H(\eta+1)^{-1} \\
&\overset{(6.10)}{\leq} \frac{1}{2}\min\{\widetilde{\operatorname{diam}}(r),\widetilde{\operatorname{diam}}(t)\} - \eta \max_{u\in\{r',s',t'\}} \widetilde{\operatorname{diam}}(u) \\
&\leq \frac{1}{2}\eta\widetilde{\operatorname{dist}}(r,t) - \eta \max_{u\in\{r',s',t'\}} \widetilde{\operatorname{diam}}(u) \\
&= \frac{1}{2}\eta\operatorname{dist}(B_r,B_t) - \eta \max_{u\in\{r',s',t'\}} \widetilde{\operatorname{diam}}(u) \\
&\leq \eta\max\{\operatorname{dist}(B_{r'},B_{s'}),\operatorname{dist}(B_{s'},B_{t'})\} + \eta\operatorname{diam}(B_{s'}) - \eta \max_{u\in\{r',s',t'\}} \widetilde{\operatorname{diam}}(u) \\
&\leq \eta\max\{\operatorname{dist}(B_{r'},B_{s'}),\operatorname{dist}(B_{s'},B_{t'})\} \\
&= \eta\max\{\widetilde{\operatorname{dist}}(r',s'),\widetilde{\operatorname{dist}}(s',t')\},
\end{aligned}$$

i.e., 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 6.26 *Lemma 6.25 proves that Construction 6.20 (→ cluster tree) combined with Construction 6.22 (→ 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 reasonable triangulations this ratio depends polynomially on $\#\mathcal{I}$ such that the logarithm of the ratio is proportional to $\log(\#\mathcal{I})$.*

Remark 6.27 (Admissibility for $\mathcal{H}^2$-matrices) *The results of Lemma 6.25 depend on the admissibility condition (2.3). In the context of $\mathcal{H}^2$-matrices the stronger admissibility condition*

$$\max\{\operatorname{diam}(\tau),\operatorname{diam}(\sigma)\} \leq \eta\operatorname{dist}(\tau,\sigma) \tag{6.16}$$

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.

6.4 Exercises

6.4.1 Theory

Exercise 23 *Let $T_{\mathcal{I}}$ be a cluster tree. Prove for all $i \in \mathbb{N}_0$*

$$\mathcal{I} = T_{\mathcal{I}}^{(i)} \cup \mathcal{L}(T_{\mathcal{I}}, i-1) \cup \ldots \cup \mathcal{L}(T_{\mathcal{I}}, 0).$$

Chapter 7

$\mathcal{H}^2$-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 [25], where the approximation was based on Taylor expansions. We will use the construction described in [13, 6], which is based on interpolation.

7.1 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

$$\tilde{g}^{t,s}(x,y) := \sum_{\nu\in K} g(x^t_\nu, y)\mathcal{L}^t_\nu(x)$$

for admissible cluster pairs (t,s) (we assume that $\operatorname{diam}(Q^t) \le \operatorname{diam}(Q^s)$ 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

$$\begin{aligned}
\tilde{G}^{t,s}_{ij} &= \int_\Omega \varphi_i(x) \int_\Omega \tilde{g}^{t,s}(x,y)\varphi_j(y)\,\mathrm{d}y\,\mathrm{d}x \\
&= \sum_{\nu\in K} \int_\Omega \varphi_i(x) \int_\Omega g(x^t_\nu,y)\mathcal{L}^t_\nu(x)\varphi_j(y)\,\mathrm{d}y\,\mathrm{d}x \\
&= \sum_{\nu\in K} \left(\int_\Omega \varphi_i(x)\mathcal{L}^t_\nu(x)\,\mathrm{d}x\right)\left(\int_\Omega \varphi_j(y) g(x^t_\nu,y)\,\mathrm{d}y\right) \\
&= \left(A^{t,s}{B^{t,s}}^\top\right)_{ij}
\end{aligned}$$

for all $i \in \hat{t}$ and $j \in \hat{s}$, where

$$A^{t,s}_{i\nu} = \int_\Omega \varphi_i(x)\mathcal{L}^t_\nu(x)\,\mathrm{d}x \quad\text{and}\quad B^{t,s}_{j\nu} = \int_\Omega \varphi_j(y) g(x^t_\nu,y)\,\mathrm{d}y.$$

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

$$V^t_{i\nu} := \int_\Omega \varphi_i(x) \mathcal{L}^t_\nu(x) \, \mathrm{d}x \tag{7.1}$$

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

$$\sum_{s \in R^t} A^{t,s} B^{t,s\top} = \sum_{s \in R^t} V^t B^{t,s\top} = V^t \left(\sum_{s \in R^t} B^{t,s\top} \right),$$

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 $T^{t',t}_{\nu'\nu} \in \mathbb{R}$ such that

$$\mathcal{L}^t_\nu = \sum_{\nu' \in K} T^{t',t}_{\nu'\nu} \mathcal{L}^{t'}_{\nu'} \tag{7.2}$$

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 $T^{t',t}_{\nu'\nu}$ is especially simple: they are given by

$$T^{t',t}_{\nu'\nu} = \mathcal{L}^t_\nu(x^{t'}_{\nu'}). \tag{7.3}$$

For each index $i \in \hat{t}$, we can find a $t' \in \operatorname{sons}(t)$ with $i \in \hat{t}'$, and equation (7.2) implies

$$V^t_{i\nu} = \int_\Omega \varphi_i(x) \mathcal{L}^t_\nu(x) \, \mathrm{d}x = \sum_{\nu' \in K} T^{t',t}_{\nu'\nu} \int_\Omega \varphi_i(x) \mathcal{L}^{t'}_{\nu'}(x) \, \mathrm{d}x = \sum_{\nu' \in K} T^{t',t}_{\nu'\nu} V^{t'}_{i\nu'} = (V^{t'} T^{t',t})_{i\nu}. \tag{7.4}$$

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 $(V^t y^t)_i = (V^{t'} T^{t',t} y^t)_i$. So instead of computing $V^t u^t$ directly, we can compute $T^{t',t} y^t$ for all sons $t' \in \operatorname{sons}(t)$, and this will only require $\mathcal{O}(k^2)$ operations.

7.2 $\mathcal{H}^2$-matrices

7.2.1 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:

$$\tilde{g}^{t,s}(x,y) := (\mathfrak{I}_m^t \otimes \mathfrak{I}_m^s)[g](x,y) = \sum_{\nu\in K}\sum_{\mu\in K} g(x_\nu^t, x_\mu^s)\mathcal{L}_\nu^t(x)\mathcal{L}_\mu^s(y). \tag{7.5}$$

Discretizing this approximation of the kernel, we find

$$\begin{aligned}\tilde{G}_{ij} &:= \int_\Omega \varphi_i(x)\int_\Omega \tilde{g}^{t,s}(x,y)\varphi_j(y)\,\mathrm{d}y\,\mathrm{d}x \\ &= \sum_{\nu\in K}\sum_{\mu\in K} g(x_\nu^t, x_\mu^s)\underbrace{\left(\int_\Omega \varphi_i(x)\mathcal{L}_\nu^t(x)\,\mathrm{d}x\right)}_{=V_{i\nu}^t}\underbrace{\left(\int_\Omega \varphi_j(y)\mathcal{L}_\mu^s(y)\,\mathrm{d}y\right)}_{=V_{j\mu}^s} = V^t S^{t,s} {V^s}^\top \end{aligned}\tag{7.6}$$

for $i \in \hat{t}$ and $j \in \hat{s}$, where $S^{t,s} \in \mathbb{R}^{K\times K}$ is defined by

$$S_{\nu\mu}^{t,s} := g(x_\nu^t, x_\mu^s). \tag{7.7}$$

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 $k := \#K$).

Definition 7.1 (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 (7.1) form a constant-order cluster basis.

Definition 7.2 (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*

$$\begin{aligned}\mathcal{H}(T_{\mathcal{I}\times\mathcal{J}}, V, W) := \{M \in \mathbb{R}^{\mathcal{I}\times\mathcal{J}} \mid\ & \textit{for all admissible } (t,s) \in \mathcal{L}(T_{\mathcal{I}\times\mathcal{J}}), \\ & \textit{there is } S^{t,s} \in \mathbb{R}^{K^t\times K^s} \textit{ with } M|_{\hat{t}\times\hat{s}} = V^t S^{t,s} {W^s}^\top\}.\end{aligned}$$

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 7.3 *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 $\mathbb{R}^{\mathcal{I}\times\mathcal{J}}$, but it is* not an ideal.

Due to $\operatorname{rank}(V^t S^{t,s} {W^s}^\top) \le \operatorname{rank}(S^{t,s}) \le \min\{\#K^t, \#K^s\}$, 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 (7.5) in each admissible block will lead to blocks satisfying (7.6), this matrix is a uniform $\mathcal{H}$-matrix of constant order with cluster bases defined by (7.1) and coefficient matrices defined by (7.7).

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 7.4 (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 $\# R^t \le C_{\mathrm{sp}}$. 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 7.5 (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 that if a leaf $t \times s$ of the block cluster tree $T_{\mathcal{I}\times\mathcal{I}}$ is not admissible, then t and s are leaves of the cluster tree $T_{\mathcal{I}}$ (cf. Exercise 8). 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 $t \times s$ be one of these leaf nodes. By assumption, t and s must be leaf clusters of $T_{\mathcal{I}}$, so we have $\max\{\#\hat{t}, \#\hat{s}\} \le n_{\min}$, and the multiplication by the corresponding full matrix can be completed in $\mathcal{O}(n_{\min}^2)$ operations. ■

In typical applications, we have $\# T_{\mathcal{I}} \le n$, 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.

7.2.2 Nested cluster bases

Definition 7.6 (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 $t' \in \mathrm{sons}(t)$, there is a* transfer matrix *$T^{t',t} \in \mathbb{R}^{K^{t'} \times K^t}$ satisfying*
$$(V^t y^t)_i = (V^{t'} T^{t',t} y^t)_i \tag{7.8}$$
for all vectors $y^t \in \mathbb{R}^{K^t}$ and all indices $i \in \hat{t}'$.

Example 7.7 (Two sons) *If we assume that $\mathrm{sons}(t) = \{t_1, t_2\}$ with $t_1 \ne t_2$, we have that $\hat{t} = \hat{t}_1 \dot\cup \hat{t}_2$ holds, and (7.8) takes the form*
$$V^t = \begin{pmatrix} V^{t_1} T^{t_1,t} \\ V^{t_2} T^{t_2,t} \end{pmatrix} = \begin{pmatrix} V^{t_1} & \\ & V^{t_2} \end{pmatrix} \begin{pmatrix} T^{t_1,t} \\ T^{t_2,t} \end{pmatrix}.$$
Applying this equation recursively, we find
$$\mathrm{range}(V^t|_{\hat{t}' \times K^t}) \subseteq \mathrm{range}(V^{t'})$$
for all clusters $t' \in T_{\mathcal{I}}$ with $\hat{t}' \subseteq \hat{t}$.

Definition 7.8 ($\mathcal{H}^2$-matrix) *A uniform $\mathcal{H}$-matrix whose column and row cluster basis are nested is called an $\mathcal{H}^2$-*matrix.

Due to equation (7.4), the cluster basis defined by (7.1) 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 (7.8) to the cluster $t' \in \operatorname{sons}(t)$ with $i \in \hat{t}'$:

$$\begin{aligned} y_i &= \sum_{s \neq t, i \in \hat{s}} (V^s y^s)_i + (V^t y^t)_i = \sum_{s \neq t, i \in \hat{s}} (V^s y^s)_i + (V^{t'} T^{t',t} y^t)_i \\ &= \sum_{s \neq t, s \neq t', i \in \hat{s}} (V^s y^s)_i + V^{t'} (y^{t'} + T^{t',t} y^t)_i. \end{aligned}$$

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

$$\hat{y}^s := \begin{cases} 0 & \text{if } s = t \\ y^s + T^{s,t} y^t & \text{if } s \in \operatorname{sons}(t) \\ y^s & \text{otherwise.} \end{cases}$$

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 7.9 (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 $T^{t',t}$ for all of its sons t' 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. ■

Remark 7.10 (Complexity of the forward transformation) *Obviously, we can treat the forward transformation by an identical argument and reach an identical bound for its the complexity.*

Remark 7.11 *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 [31, 32, 8].*

7.2.3 Implementation

The structure used for the representation of a cluster basis is similar to that used for clusters:

Implementation 7.12 (`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 $T^{t',t}$. 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 7.13 *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 7.14 (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.

7.3 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 7.15 (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 7.16 (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 7.17 (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: Exercise 26. ■

Lemma 7.18 (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

$$\|M - V^t S_M (W^s)^\top\|_F \le \|M - V^t S (W^s)^\top\|_F$$

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

$$0 = Df(S^*)\cdot R = \langle V^t S^* (W^s)^\top, V^t R (W^s)^\top\rangle_F - \langle M, V^t R (W^s)^\top\rangle_F \tag{7.9}$$

for all $R \in \mathbb{R}^{\hat{t} \times \hat{s}}$, i.e.,

$$\begin{aligned}\langle S_M, R \rangle_F &= \langle (V^t)^\top M W^s, R \rangle_F = \langle M, V^t R (W^s)^\top \rangle_F \overset{(7.9)}{=} \langle V^t S^* (W^s)^\top, V^t R (W^s)^\top \rangle_F \\ &= \langle S^*, \underbrace{(V^t)^\top V^t}_{=I} R \underbrace{(W^s)^\top W^s}_{=I} \rangle_F = \langle S^*, R \rangle_F.\end{aligned}$$

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);
}
```

7.4 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. [5, 7, 2]).

7.4.1 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 7.19 (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*

$$\|M - V^t S_M (W^s)^\top\|_F^2 \le \|M - V^t (V^t)^\top M\|_F^2 + \|M^\top - W^s (W^s)^\top M^\top\|_F^2.$$

Proof: Let $B, C \in \mathbb{R}^{\hat{t}\times\hat{s}}$. We start by observing

$$\langle B - V^t(V^t)^\top B, V^t(V^t)^\top C\rangle_F = \langle V^t(V^t)^\top B - V^t \underbrace{(V^t)^\top V^t}_{=I}(V^t)^\top B, C\rangle_F = 0. \tag{7.10}$$

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 (7.10), we can again use Pythagoras' equation to get

$$\|B\|_F^2 = \|B - V^t(V^t)^\top B\|_F^2 + \|V^t(V^t)^\top B\|_F^2 \ge \|V^t(V^t)^\top B\|_F^2$$

and therefore

$$\|V^t(V^t)^\top(M - MW^s(W^s)^\top)\|_F^2 \le \|M - MW^s(W^s)^\top\|_F^2.$$

Observing $\|M - MW^s(W^s)^\top\|_F = \|M^\top - W^s(W^s)^\top M^\top\|_F$ concludes the proof. ■

Lemma 7.20 (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 (7.10), 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 $\|(V^t)^\top M\|_F$.

7.4.2 Construction of a cluster basis for one cluster

We recall Lemma 5.5 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 $\Sigma = \operatorname{diag}(\sigma_1, \dots, \sigma_n) \in \mathbb{R}^{n\times n}$, where $\sigma_1 \ge \sigma_2 \ge \dots \ge \sigma_n \ge 0$. 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 $i \in \{1, \dots, n\}$, $j \in \{1, \dots, k\}$ 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

$$I^{n,k}(I^{n,k})^\top_{ij} = \begin{cases} \delta_{ij} & \text{if } i \leq k_t \\ 0 & \text{otherwise,} \end{cases}$$

we get

$$\|M - V^t(V^t)^\top M\|_F^2 = \sum_{i=k+1}^{n} \sigma_i^2.$$

We note that

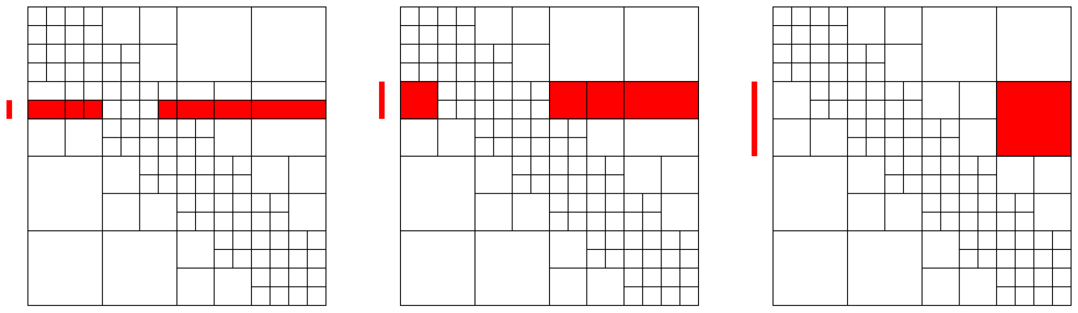
$$MM^\top = V\Sigma U^\top U \Sigma^\top V^\top = V\Sigma^2 V^\top$$

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

$$\sum_{s \in R_+^t} \|(V^t)^\top M|_{\hat{t}\times\hat{s}}\|_F^2. \tag{7.11}$$

As we have seen before, this problem can be solved by computing the eigenvectors and eigenvalues of the Gram matrix

$$G := \sum_{s \in R_+^t} M|_{\hat{t}\times\hat{s}} M|_{\hat{t}\times\hat{s}}^\top. \tag{7.12}$$

7.4.3 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 $t' \in \mathrm{sons}(t)$ and have to ensure that there are matrices $T^{t',t} \in \mathbb{R}^{k^{t'} \times k}$ such that

$$V_i^t = (V^{t'} T^{t',t})_i$$

holds for all $t' \in \mathrm{sons}(t)$ and $i \in \hat{t}$. In order to simplify the presentation, we will only consider the case that t has two sons, i.e., $\mathrm{sons}(t) = \{t_1, t_2\}$. Then the condition (7.8) can be written in the form

$$V^t = \begin{pmatrix} V^{t_1} T^{t_1,t} \\ V^{t_2} T^{t_2,t} \end{pmatrix}$$

and instead of finding V^t maximizing (7.11), we have to find $T^{t_1,t}$ and $T^{t_2,t}$ maximizing

$$\sum_{s \in R_+^t} \left\| \begin{pmatrix} V^{t_1} T^{t_1,t} \\ V^{t_2} T^{t_2,t} \end{pmatrix}^\top \begin{pmatrix} M|_{\hat{t}_1 \times \hat{s}} \\ M|_{\hat{t}_2 \times \hat{s}} \end{pmatrix} \right\|_F^2 = \sum_{s \in R_+^t} \left\| \begin{pmatrix} T^{t_1,t} \\ T^{t_2,t} \end{pmatrix}^\top \begin{pmatrix} (V^{t_1})^\top M|_{\hat{t}_1 \times \hat{s}} \\ (V^{t_2})^\top M|_{\hat{t}_2 \times \hat{s}} \end{pmatrix} \right\|_F^2$$

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

$$\sum_{s \in R_+^t} \|(V^t)^\top M|_{\hat{t} \times \hat{s}}\|_F^2 = \sum_{s \in R_+^t} \left\| \begin{pmatrix} T^{t_1,t} \\ T^{t_2,t} \end{pmatrix}^\top \begin{pmatrix} \widehat{M}^{t_1,s} \\ \widehat{M}^{t_2,s} \end{pmatrix} \right\|_F^2 .$$

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

$$I = (V^t)^\top V^t = \begin{pmatrix} V^{t_1} T^{t_1,t} \\ V^{t_2} T^{t_2,t} \end{pmatrix}^\top \begin{pmatrix} V^{t_1} T^{t_1,t} \\ V^{t_2} T^{t_2,t} \end{pmatrix} = \begin{pmatrix} T^{t_1,t} \\ T^{t_2,t} \end{pmatrix}^\top \begin{pmatrix} (V^{t_1})^\top V^{t_1} T^{t_1,t} \\ (V^{t_2})^\top V^{t_2} T^{t_2,t} \end{pmatrix} = \begin{pmatrix} T^{t_1,t} \\ T^{t_2,t} \end{pmatrix}^\top \begin{pmatrix} T^{t_1,t} \\ T^{t_2,t} \end{pmatrix}.$$

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

   $$\widehat{M}^{t,s} := \begin{pmatrix} T^{t_1,t} \\ T^{t_2,t} \end{pmatrix}^\top \begin{pmatrix} \widehat{M}^{t_1,s} \\ \widehat{M}^{t_2,s} \end{pmatrix} = (T^{t_1,t})^\top \widehat{M}^{t_1,s} + (T^{t_2,t})^\top \widehat{M}^{t_2,s}$$

   for all $s \in R_+^t$.

7.4.4 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

$$M|_{\hat{t}\times\hat{s}} M|_{\hat{t}\times\hat{s}}^\top = A|_{\hat{t}\times k} B^\top B A|_{\hat{t}\times k}^\top = A|_{\hat{t}\times k} G_b A|_{\hat{t}\times k}^\top. \qquad (7.13)$$

If we have prepared the matrix

$$G_b := B^\top B,$$

in advance, we can compute the product (7.13) 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 $\mathcal{O}(nk^2_\mathcal{H}p)$ operations.

7.5 Implementation

Implementation 7.21 (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 7.18). 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.

7.6 Exercises

7.6.1 Theory

Exercise 24 (Forward transformation) *Derive a fast algorithm for computing the forward transformation based on the property* (7.8) *of nested cluster bases.*

Prove that it has the same complexity as the backward transformation (cf. Lemma 7.9).

Exercise 25 (Error bound for the symmetric kernel approximation) *Assume that*

$$\operatorname{diam}(Q^t \times Q^s) \le \eta \operatorname{dist}(Q^t, Q^s) \tag{7.14}$$

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 3.2.4.

Exercise 26 (Frobenius and matrix product) *Prove Lemma 7.17.*

7.6.2 Practice

Exercise 27 (Forward transformation) *Implement the algorithm from Exercise 24 as a function*

```
void
forward_clusterbasis(pclusterbasis cb, const double *x);
```

that fills the entries `cb->xt` *of this* `clusterbasis` *structure and all its descendants with the coefficient vectors* $x^t = V^t x|_{\hat{t}}$.

Hint: *See the procedure* `backward_clusterbasis` *in the source file* `clusterbasis.c`.

Exercise 28 (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* (2.3) *by the strong admissibility condition* (7.14).

Modify the function `build_supermatrix_from_cluster2` *from Exercise 8 in such a way that it uses the strong admissibility condition* (7.14) *instead of the standard admissibility.*

Exercise 29 ($\mathcal{H}^2$-matrix) *Modify* `build_supermatrix_from_cluster2` *from Exercise 28 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.*

Exercise 30 (Coefficient matrices) *In Exercise 29, we have created an $\mathcal{H}^2$-matrix structure. In order to get a useful matrix, we have to initialize the coefficient matrices $S^{t,s}$, i.e., the field* `S` *in each of the* `uniformmatrix` *structures (cf.* (7.7)*) Add a function*

```
static void
interpolate_kernel(pclusterbasis row, pclusterbasis column,
                   pcurvebemfactory bfactory,
                   double *S, int ldS);
```

to the module `curvebem.c` *that initializes the coefficient matrix* `S` *corresponding to the clusters* `row->t` *and* `column->t` *for the kernel function from Subsection 3.3.*

Hint: *The fields* `row->t` *and* `column->t` *give us the clusters corresponding to a row and a column, so we can easily access the bounding boxes needed in the computation of the interpolation points.*

Test: *Compile the program* `example_h2` *by executing*

```
make -f Makefile.sun example_h2
```

in a shell and run it with 1024 *basis functions and an interpolation order of* 3. *The program should reach a relative approximation error of less than* 0.0003.

Chapter 8

Matrix Equations

In this chapter we consider matrix equations of the form

$$\begin{aligned}
&\text{(Lyapunov)} && AX + XA^T + C = 0, && (8.1)\\
&\text{(Sylvester)} && AX - XB + C = 0, && (8.2)\\
&\text{(Riccati)} && AX + XA^T - XFX + C = 0, && (8.3)
\end{aligned}$$

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.

8.1 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 4.3 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.

8.2 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 (8.2) it is necessary and sufficient that $\sigma(A) \cap \sigma(B) = \emptyset$. We demand a slightly stronger condition, namely

$$\sigma(A) < \sigma(B), \qquad (8.4)$$

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 8.1 *Let $A \in \mathbb{R}^{n\times n}, B \in \mathbb{R}^{m\times m}$ and let (8.4) hold. Then the matrix*

$$X := \int_0^\infty \exp(tA)C\exp(-tB)\,\mathrm{d}t \tag{8.5}$$

solves (8.2).

Proof: Due to (8.4) the function $t \mapsto \|\exp(tA)\|\|\exp(-tB)\|$ decays exponentially so that the matrix X is well defined. Now we can simply calculate

$$\begin{aligned} AX - XB &= \int_0^\infty \left(A\exp(tA)C\exp(-tB) - \exp(tA)C\exp(-tB)B\right)\,\mathrm{d}t \\ &= \int_0^\infty \frac{\partial}{\partial_t}\exp(tA)C\exp(-tB)\,\mathrm{d}t \\ &= \lim_{t\to\infty}\exp(tA)C\exp(-tB) - \exp(0\cdot A)C\exp(-0\cdot B) \\ &= -C. \end{aligned}$$

■

Since the integrand in (8.5) 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 [33] as done in [17] that gives k weights w_j and points t_j so that the matrix

$$X_k := \sum_{j=1}^k w_j \exp(t_jA)C\exp(-t_jB) \tag{8.6}$$

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 (8.6) is not straight-forward, since it involves the matrix exponential.

8.3 Existence of $\mathcal{H}$-matrix Solutions

Formula (8.6) 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 $\exp(t_jA)$ and $\exp(-t_jB)$ 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 6.18 proves

$$A_jCB_j \in \mathcal{H}(T,\ C_{\mathrm{id}}^2C_{\mathrm{sp}}^2(p+1)^2\max\{k_A, k_B, k_C\})$$

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 [11, 17] and briefly in the outlook in chapter 9.3.2. 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 4.3, 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 $\tilde{k} = \mathcal{O}(\log(\varepsilon)^2 kp)$.

8.4 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 (8.6). 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 [4] or by a Dunford-Cauchy representation that involves the resolvents [11]. For each quadrature point we have to compute at least once a matrix exponential, i.e., the complexity is at least $\mathcal{O}(n \log^4)$ 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 [22]. 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 [29]. 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 [17]. 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 initital 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 $\mathcal{O}(n\log(n)^3)$.

8.5 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

$$b = \bigotimes_{i=1}^{d} b_i, \qquad b_i \in \mathbb{R}^n, \quad b[j_1, \dots, j_d] = \prod_{i=1}^{d} b_i[j_i] \quad \text{for } j \in \{1, \dots, n\}^d.$$

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 [15]. 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 9

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.

9.1 Adaptive Arithmetics and Adaptive Refinement

9.1.1 Rank-adaptive Truncation

In Chapter 5 we have defined the formatted arithmetics of $\mathcal{R}(k)$-matrices and $\mathcal{H}$-matrices based on a fixed given rank k in the target matrix. E.g., for the addition of two hierarchical matrices we have truncated the exact sum in each `rkmatrix` block to rank k by the projection $\mathcal{T}_k$. An alternative truncation operator $\mathcal{T}_\varepsilon$ can be defined by

$$\mathcal{T}_\varepsilon(M) := \operatorname{argmin}\left\{\operatorname{rank}(R) \;\middle|\; \frac{\|R-M\|_2}{\|M\|_2} \le \varepsilon\right\}.$$

Again, the reduced singular value decomposition (rSVD) allows for an efficient computation of the minimizer. The parameter ε can be regarded as the desired accuracy, e.g., $\varepsilon = 10^{-16}$ is the double precision machine accuracy, $\varepsilon = 10^{-8}$ the single precision accuracy and any larger ε yields an approximation of lower quality.

The corresponding truncation $\mathcal{T}_\varepsilon$ of hierarchical matrices leads to the so-called adaptive arithmetics (see [14]), where the rank is not a priori fixed. This is especially interesting in the context of the inversion of stiffness matrices of partial differential operators where the sparsity of the matrix to be inverted can be kept longer than for fixed rank. Also, few blocks can have a considerably larger rank if needed and thus allow for the treatment of matrices that would not fit into the hierarchical matrix format for (constant) fixed rank in each block.

9.1.2 Adaptive Grid-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. This topic is partially covered by [18].

9.2 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.

9.3 Applications of Hierarchical Matrices

9.3.1 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 [5] 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.

9.3.2 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 4.3 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 $t_j = j\delta$, $j = 1, \dots, N$, 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 [11] under the assumption that the resolvents $(A - \lambda I)^{-1}$ can be approximated in the $\mathcal{H}$-matrix format, cf. Chapter 4.3. For the second question there is no definitive answer; often the standard scaling and squaring strategy proposed in [4] does the job very well but for a more general survey the reader is referred to [28].

For other matrix functions ([10],[12]) 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] Mario Bebendorf and Wolfgang Hackbusch. Existence of $\mathcal{H}$-matrix approximants to the inverse FE-matrix of elliptic operators with L^∞-coefficients. *Numerische Mathematik*, 95:1–28, 2003.

[2] Steffen Börm. $\mathcal{H}^2$-matrices — multilevel methods for the approximation of integral operators. *Comput. Visual. Sci.*, 7:173–181, 2004.

[3] Steffen Börm and Lars Grasedyck. Low-rank approximation of integral operators by interpolation. *Computing*, 72:325–332, 2004.

[4] Steffen Börm, Lars Grasedyck, and Wolfgang Hackbusch. Introduction to hierarchical matrices with applications. *Engineering Analysis with Boundary Elements*, 27:405–422, 2003.

[5] Steffen Börm and Wolfgang Hackbusch. Data-sparse approximation by adaptive $\mathcal{H}^2$-matrices. *Computing*, 69:1–35, 2002.

[6] Steffen Börm and Wolfgang Hackbusch. $\mathcal{H}^2$-matrix approximation of integral operators by interpolation. *Applied Numerical Mathematics*, 43:129–143, 2002.

[7] Steffen Börm and Wolfgang Hackbusch. Approximation of boundary element operators by adaptive $\mathcal{H}^2$-matrices. *Foundations of Computational Mathematics*, 312:58–75, 2004.

[8] Steffen Börm, Maike Löhndorf, and Jens Markus Melenk. Approximation of integral operators by variable-order interpolation. *Numerische Mathematik*, 99(4):605–643, 2005.

[9] C. F. Van Loan G. H. Golub. *Matrix Computations.* Johns Hopkins University Press, London, 1996.

[10] Ivan Gavrilyuk, Wolfgang Hackbusch, and Boris Khoromskij. Data-sparse approximation to operator-valued functions of elliptic operator. Technical Report 54, Max Planck Institute for Mathematics in the Sciences, 2002. To appear in Mathematics of Computation.

[11] Ivan Gavrilyuk, Wolfgang Hackbusch, and Boris Khoromskij. $\mathcal{H}$-matrix approximation for the operator exponential with applications. *Numerische Mathematik*, 92:83–111, 2002.

[12] Ivan Gavrilyuk, Wolfgang Hackbusch, and Boris Khoromskij. Data-sparse approximation of a class of operator-valued functions. Technical Report 20, Max Planck Institute for Mathematics in the Sciences, 2003. To appear in Mathematics of Computation.

[13] Klaus Giebermann. Multilevel approximation of boundary integral operators. *Computing*, 67:183–207, 2001.

[14] Lars Grasedyck. *Theorie und Anwendungen Hierarchischer Matrizen.* PhD thesis, Universität Kiel, 2001.

[15] Lars Grasedyck. Existence and computation of low Kronecker-rank approximations for large linear systems of tensor product structure. *Computing*, 72:247–265, 2004.

[16] Lars Grasedyck and Wolfgang Hackbusch. Construction and arithmetics of $\mathcal{H}$-matrices. *Computing*, 70(4):295–334, 2003.

[17] 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.

[18] Lars Grasedyck, Wolfgang Hackbusch, and Sabine LeBorne. Adaptive refinement and clustering of $\mathcal{H}$-matrices. Technical Report 106, Max Planck Institute of Mathematics in the Sciences, 2001.

[19] Leslie Greengard and Vladimir Rokhlin. A fast algorithm for particle simulations. *Journal of Computational Physics*, 73:325–348, 1987.

[20] M. Güter and K.-O. Widman. The green function for uniformly elliptic equations. *Manuscripta Mathematica*, 37:303–342, 1982.

[21] Wolfgang Hackbusch. *Elliptic Differential Equations. Theory and Numerical Treatment.* Springer-Verlag Berlin, 1992.

[22] Wolfgang Hackbusch. *Iterative Solution of Large Sparse Systems.* Springer-Verlag New York, 1994.

[23] Wolfgang Hackbusch. A sparse matrix arithmetic based on $\mathcal{H}$-matrices. Part I: Introduction to $\mathcal{H}$-matrices. *Computing*, 62:89–108, 1999.

[24] Wolfgang Hackbusch and Boris Khoromskij. A sparse $\mathcal{H}$-matrix arithmetic: General complexity estimates. *J. Comp. Appl. Math.*, 125:479–501, 2000.

[25] 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.

[26] Wolfgang Hackbusch and Zenon Paul Nowak. On the fast matrix multiplication in the boundary element method by panel clustering. *Numerische Mathematik*, 54:463–491, 1989.

[27] G. Meinardus. *Approximation of Functions: Theory and Numerical Methods.* Springer-Verlag New York, 1967.

[28] 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.

[29] 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.

[30] Vladimir Rokhlin. Rapid solution of integral equations of classical potential theory. *Journal of Computational Physics*, 60:187–207, 1985.

[31] Stefan Sauter. Variable order panel clustering (extended version). Technical Report 52, Max-Planck-Institut für Mathematik, Leipzig, Germany, 1999.

[32] Stefan Sauter. Variable order panel clustering. *Computing*, 64:223–261, 2000.

[33] F. Stenger. *Numerical methods based on Sinc and analytic functions.* Springer, New York, 1993.