

## Chapter 2

# Hierarchical Matrices

In this chapter let  $T_I$  and  $T_J$  be binary cluster trees for the index sets  $I$  and  $J$ , respectively. A generalization to arbitrary cluster trees is possible; see Sect. 4.5 for nested dissection cluster trees, which are ternary. The block cluster tree  $T_{I \times J}$  is assumed to be generated using a given admissibility condition as described in the previous chapter. We will define the set of hierarchical matrices originally introduced by Hackbusch [127] and Hackbusch and Khoromskij [133, 132]; see also [128]. The elements of this set can be stored with logarithmic-linear complexity and provide data-sparse representations of fully populated matrices. Additionally, combining the hierarchical partition and the efficient structure of low-rank matrices, an approximate algebra can be defined which is based on divide-and-conquer versions of the usual block operations. The efficient replacements for matrix addition and matrix multiplication can be used to define substitutes for higher level matrix operations such as inversion,  $LU$  factorization, and  $QR$  factorization.

After the definition of hierarchical matrices in Sect. 2.1, we consider the multiplication of such matrices by a vector in Sect. 2.2. It will be seen that this can be done with logarithmic-linear complexity. In Sect. 2.3 we describe how to perform this multiplication on a parallel computer. Matrix operations such as addition and multiplication and relations between local and global norm estimates were investigated in detail in [116]. We review the results and improve some of the estimates in Sect. 2.4, Sect. 2.5, and Sect. 2.7, since these results will be important for higher matrix operations. Furthermore, we adapt the proofs to our way of clustering and present an improved addition algorithm which can be shown to preserve positivity. An accelerated matrix multiplication can be defined (see Sect. 2.7.4) if one of the factors is semi-separable. In Sect. 2.6 we analyze a technique for reducing the computational costs of  $\mathcal{H}$ -matrix approximants by unifying neighboring blocks. In Sect. 2.8, Sect. 2.9, and Sect. 2.10 the  $\mathcal{H}$ -matrix inversion,  $LU$ , and  $QR$  factorization are presented. In Sect. 2.11 we point out the similarities of  $\mathcal{H}^2$ -matrices with fast multipole methods. Finally, in Sect. 2.12 we investigate the required accuracy of  $\mathcal{H}$ -matrix approximants if they are to be used for preconditioning. It will be seen that low-precision approximations will be sufficient to guarantee a bounded number of preconditioned iterations.  $\mathcal{H}$ -matrix preconditioners can be constructed in a

purely algebraic way using the hierarchical inverse or the hierarchical  $LU$  decomposition.

Since we do not want to exploit properties of the underlying operator at this point, the complexity is estimated in terms of the maximum rank among the blocks in the partition. The size of this rank will be analyzed depending on the prescribed accuracy in the second part of this book when more properties of the underlying operator can be accessed. If the blockwise rank is assumed to scale logarithmically with the number of degrees of freedom, all presented operations can be seen to have logarithmic-linear complexity.

## 2.1 The Set of Hierarchical Matrices

**Definition 2.1.** The set of **hierarchical matrices** on the block cluster tree  $T_{I \times J}$  with admissible partition  $P := \mathcal{L}(T_{I \times J})$  and blockwise rank  $k$  is defined as

$$\mathcal{H}(T_{I \times J}, k) = \{A \in \mathbb{C}^{I \times J} : \text{rank } A_b \leq k \text{ for all admissible } b \in P\}.$$

For the sake of brevity, elements from  $\mathcal{H}(T_{I \times J}, k)$  will often be called  $\mathcal{H}$ -matrices.

*Remark 2.2.* For an efficient treatment of admissible blocks the outer-product representation from Sect. 1.1.1 should be used. Additionally, it is advisable not to use the maximum rank  $k$  but the actual rank of the respective block as the number of rows and columns. Storing non-admissible blocks entrywise will increase the efficiency.

*Example 2.3.* The left picture of Fig. 2.1 shows an  $\mathcal{H}$ -matrix approximant for the matrix  $a_{ij} = (|i - j| + 1)^{-1}$ ,  $i, j = 1, \dots, n$ . The right picture represents an approximant for the *Hilbert matrix*  $h_{ij} = (i + j - 1)^{-1}$ ,  $i, j = 1, \dots, n$ . Depending on the kind of “singularity”, the admissibility conditions from the Examples 1.13 and 1.12 have to be used. For the Hilbert matrix an approximant with accuracy  $\varepsilon = 10^{-4}$  can be found which for  $n = 1000$  reduces the amount of storage to 3.2% and for  $n = 100000$  to 0.32%.

A few easy consequences are gathered in the following two lemmas.

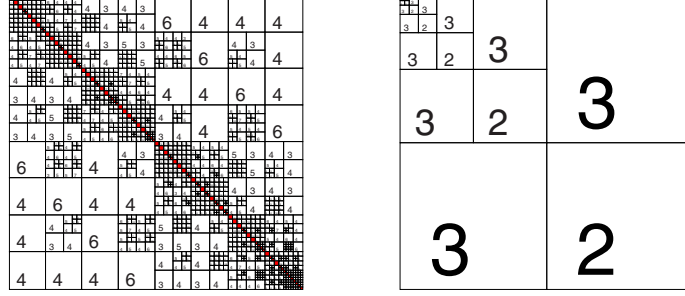
**Lemma 2.4.** Let  $A \in \mathcal{H}(T_{I \times J}, k)$ . Then

- (i) any submatrix  $A_b$ ,  $b \in T_{I \times J}$ , belongs to  $\mathcal{H}(T_b, k)$ ;
- (ii) the transpose  $A^T$  and the Hermitian transpose  $A^H$  belong to  $\mathcal{H}(T_{J \times I}, k)$  provided that the admissibility condition is symmetric; i.e., any block  $s \times t$  is admissible if  $t \times s$  is admissible.

We define the set

$$\pi_I := \{t \in T_I : \exists s \in T_J \text{ such that } t \times s \in P \text{ and } \forall t' \subset t \text{ and } \forall s' \in T_J : t' \times s' \notin P\},$$

which is the finest partition of  $I$  made from clusters appearing in the partition  $P$ .



**Fig. 2.1**  $\mathcal{H}$ -matrices with their blockwise ranks.

**Lemma 2.5.** *Let  $D_1, D_2$  be block diagonal matrices on the tensor partitions  $\pi_I \times \pi_I$  and  $\pi_J \times \pi_J$ , respectively. Then  $D_1 A D_2 \in \mathcal{H}(T_{I \times J}, k)$  provided that  $A \in \mathcal{H}(T_{I \times J}, k)$ .*

*Proof.* Due to the assumption, each block  $t \times s \in P$  of  $D_1 A D_2$  arises from multiplying  $A_{ts}$  by  $(D_1)_{tt}$  and  $(D_2)_{ss}$ . Hence, from Theorem 1.1 we have that  $\text{rank}(D_1 A D_2)_{ts} = \text{rank}(D_1)_{tt} A_{ts} (D_2)_{ss} \leq \text{rank} A_{ts}$ .  $\square$

We have already seen in Sect. 1.1.1 that the storage requirements for an admissible block  $b = t \times s \in \mathcal{L}(T_{I \times J})$  of  $A \in \mathcal{H}(T_{I \times J}, k)$  are

$$N_{\text{st}}(A_b) \leq k(|t| + |s|).$$

A non-admissible block  $A_b, b \in P$ , is stored entrywise and thus requires  $|t||s|$  units of storage. Since  $\min\{|t|, |s|\} \leq n_{\min}$  (see Remark 1.17) we have

$$|t||s| = \min\{|t|, |s|\} \max\{|t|, |s|\} \leq n_{\min}(|t| + |s|). \quad (2.1)$$

Hence, for storing  $A_b, b \in P$ , at most  $\max\{k, n_{\min}\}(|t| + |s|)$  units of storage are required. Using (1.36), we obtain the following theorem.

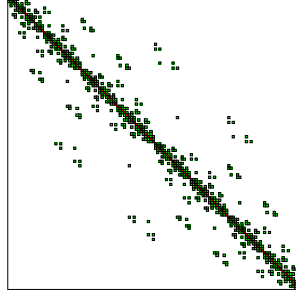
**Theorem 2.6.** *Let  $c_{\text{sp}}$  be the sparsity constant for the tree  $T_{I \times J}$ ; cf. Definition 1.35. The storage requirements  $N_{\text{st}}$  for  $A \in \mathcal{H}(T_{I \times J}, k)$  are bounded by*

$$N_{\text{st}}(A) \leq c_{\text{sp}} \max\{k, n_{\min}\} [L(T_I)|I| + L(T_J)|J|].$$

*If  $T_I$  and  $T_J$  are balanced cluster trees (cf. Definition 1.19), we have*

$$N_{\text{st}}(A) \sim \max\{k, n_{\min}\} [|I| \log |I| + |J| \log |J|].$$

**Remark 2.7.** Although  $\mathcal{H}$ -matrices are primarily aiming at dense matrices, sparse matrices  $A$  which vanish on admissible blocks are also in  $\mathcal{H}(T_{I \times J}, n_{\min})$ . Since the size of one of the clusters corresponding to non-admissible blocks is less than or equal to  $n_{\min}$ , the rank of each block  $A_b$  does not exceed  $n_{\min}$ . A deeper analysis



**Fig. 2.2** A sparse  $\mathcal{H}$ -matrix (nonzero blocks are shown).

(see Lemma 4.2) shows that finite element discretizations can actually be stored with linear complexity.

## 2.2 Matrix-Vector Multiplication

The cost of multiplying an  $\mathcal{H}$ -matrix  $A \in \mathcal{H}(T_{I \times J}, k)$  or its Hermitian transpose  $A^H$  by a vector  $x$  is inherited from the blockwise matrix-vector multiplication:

$$Ax = \sum_{t \times s \in P} A_{ts} x_s \quad \text{and} \quad A^H x = \sum_{t \times s \in P} (A_{ts})^H x_t. \quad (2.2)$$

Since each admissible block  $t \times s$  has the outer product representation  $A_{ts} = UV^H$ ,  $U \in \mathbb{C}^{t \times k}$ ,  $V \in \mathbb{C}^{s \times k}$ , at most  $2k(|t| + |s|)$  operations are required to compute the matrix-vector products  $A_{ts}x_s = UV^H x_s$  and  $(A_{ts})^H x_t = VU^H x_t$ . If  $t \times s$  is non-admissible, then  $A_{ts}$  is stored entrywise and  $\min\{|t|, |s|\} \leq n_{\min}$ . As in (2.1) we see that in this case

$$2|t||s| \leq 2n_{\min}(|t| + |s|)$$

arithmetic operations are required. The same arguments that were used for Theorem 2.6 give the following theorem.

**Theorem 2.8.** *For the number of operations  $N_{\text{MV}}$  required for one matrix-vector multiplication  $Ax$  of  $A \in \mathcal{H}(T_{I \times J}, k)$  by a vector  $x \in \mathbb{C}^J$  it holds that*

$$N_{\text{MV}}(A) \leq 2c_{\text{sp}} \max\{k, n_{\min}\} [L(T_I)|I| + L(T_J)|J|].$$

*If  $T_I$  and  $T_J$  are balanced cluster trees, we have*

$$N_{\text{MV}}(A) \sim \max\{k, n_{\min}\} [|I| \log |I| + |J| \log |J|].$$

Hence,  $\mathcal{H}$ -matrices are well suited for iterative schemes such as Krylov subspace methods (see [221]), which the matrix enters only through the matrix-vector product.

Obviously, the matrix-vector multiplication can also be done by a recursion through the block cluster tree  $T_{I \times J}$ . Since arithmetic operations are performed only on the leaves of  $T_{I \times J}$ , summing over the leaves of  $T_{I \times J}$  as it is done in (2.2) is slightly more efficient. The latter representation is also more convenient for the following parallelization of the matrix-vector multiplication.

## 2.3 Parallel Matrix-Vector Multiplication

Although the product  $Ax$  of a matrix  $A \in \mathcal{H}(T_{I \times J}, k)$  and a vector  $x \in \mathbb{C}^J$  can be done with almost linear complexity, it can still be helpful to further reduce the execution time especially if many matrix-vector products are to be computed as part of an iterative solver, for instance. The parallelization of algorithms is always a promising way to achieve this reduction provided that significant parts of the algorithm admit independent execution. The following ideas were presented in [29].

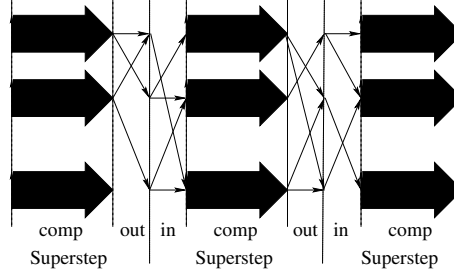
Instead of a pure matrix-vector multiplication, in this section we examine the more general update of a vector  $y \in \mathbb{C}^I$  by the operation

$$y := \alpha Ax + \beta y$$

with scalars  $\alpha, \beta \in \mathbb{C}$ . For this purpose we present two algorithms, one for distributed and the other for shared memory systems. For their description we use a unified model of a general parallel computer, a so-called **bulk synchronous parallel (BSP) computer**; see [253]. This model of a parallel system is based on three parameters: the number of processors  $p$ , the number of time steps for a global synchronization  $\ell$ , and the ratio  $g$  of the total number of operations performed on all processors and the total number of words delivered by the communication network per second. All parameters are normalized with respect to the number of time steps per second.

A BSP computation consists of single *supersteps*. Each superstep has an *input phase*, a *local computation phase* and an *output phase*; see Fig. 2.3. During the input phase, each processor receives data sent during the output phase of the previous superstep. While all processors are synchronized after each superstep, all computations within each superstep are asynchronous.

The complexity of a BSP computation can be described by the parameters  $\ell, g$ , the number of operations done on each processor, and the amount of data sent between the processors. The amount of work for a single superstep can be expressed by  $w + (h_{\text{in}} + h_{\text{out}}) \cdot g + \ell$ , where  $w$  is the maximum number of operations performed and  $h_{\text{in}}, h_{\text{out}}$  are the maximum numbers of data units received and sent by each processor, respectively. The total work of the BSP computation is the sum of the costs



**Fig. 2.3** BSP computation.

in each superstep, yielding an expression of the form  $W + H \cdot g + S \cdot \ell$ , where  $S$  is the number of supersteps.

The complexity analysis of the presented methods will be done in terms of parallel speedup and parallel efficiency.

**Definition 2.9.** Let  $t(p)$  denote the time needed by a parallel algorithm with  $p$  processors. Then

$$S(p) := \frac{t(1)}{t(p)}$$

denotes the **parallel speedup** and

$$E(p) := \frac{S(p)}{p} = \frac{t(1)}{p \cdot t(p)}$$

its **parallel efficiency**.

We assume that both vectors  $x$  and  $y$  are distributed uniformly among  $p$  processors, each holding  $|J|/p$  and  $|I|/p$  entries of  $x$  and  $y$ , respectively. By  $x_q$  and  $y_q$  we denote the part of  $x$  and  $y$  on processor  $0 \leq q < p$ . This distribution ensures optimal complexity of all vector operations.

### 2.3.1 Parallelization for Usual Matrices

As a first step towards the hierarchical matrix-vector multiplication on a parallel machine we review the ideas of the BSP algorithm for dense matrices described in [181].

Consider the case of a dense matrix  $A \in \mathbb{C}^{I \times J}$ . Let each of the  $p$  processors hold a block  $A_q$  of size  $(|I|/\sqrt{p}) \times (|J|/\sqrt{p})$  from a uniform partition of  $I \times J$ . The BSP algorithm is split into three steps. In the first step, each processor has to receive  $|J|/\sqrt{p}$  entries of  $x$  needed for the local matrix-vector multiplication, which is done in the second superstep. The resulting entries of  $y$  are afterwards sent to the corresponding

processors such that in the third step all local coefficients of  $y$  can be summed up for the final result.

---

```

procedure dense_mult( $\alpha, A_q, x, \beta, y, q$ )
  { first step }
   $y_q := \beta \cdot y_q$ ;
  send  $x_q$  to all processors sharing it;
  sync();
  { second step }
   $y'_q := \alpha A_q x_q$ ;
  send respective parts of  $y'_q$  to all processors sharing it;
  sync();
  { third step }
   $Y_q := \{\text{received vectors of local results}\}$ ;
   $y_q := y_q + \sum_{y'_j \in Y_q} y'_j$ ;
  sync();
end;

```

---

Algorithm 2.1: Dense matrix-vector multiplication.

The costs of Algorithm 2.1 are  $|I|/p + g \cdot |J|/\sqrt{p} + \ell$  for scaling  $y$  and sending  $x$  in the first step,  $|I||J|/p + g \cdot |I|/\sqrt{p} + \ell$  for the local matrix-vector product in the second step, and  $|I|/\sqrt{p} + \ell$  for the summation in the last superstep. Therefore, the total costs required to multiply a dense matrix by a vector can be estimated as

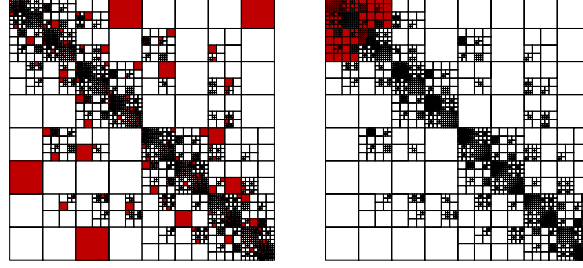
$$\mathcal{O}\left(\frac{|I||J|}{p} + \frac{|I| + |J|}{\sqrt{p}}\right) + g \cdot \mathcal{O}\left(\frac{|I| + |J|}{\sqrt{p}}\right) + 3 \cdot \ell, \quad (2.3)$$

which can be shown to be optimal with respect to computation and communication costs; cf. [181].

### 2.3.2 Non-Uniform Block Distributions

The uniform block distribution which was used in the previous section is not suitable for  $\mathcal{H}$ -matrices since the costs of a matrix-vector multiplication vary among the blocks of an  $\mathcal{H}$ -matrix due to their different sizes and their different representations. Unfortunately, changing the distribution pattern is likely to result in communication costs which are no longer optimal. For instance, the random distribution in Fig. 2.4 (left) results from applying **list scheduling**, i.e., assigning the next not yet executed job to the first idle processor. Although list scheduling guarantees an efficient local multiplication phase in the second step of Algorithm 2.1, the vector  $x$  has to be sent to all processors with communication costs of  $\mathcal{O}(|J|)$  due to the scattering of the matrix blocks across the whole  $\mathcal{H}$ -matrix. Furthermore,  $p$  vectors have to be summed up in the third step with computational costs  $\mathcal{O}(|I|)$ . Such a situation should therefore be avoided.

In order to be able to measure the communication and computation costs with respect to the vectors  $x$  and  $y$ , we introduce the *sharing constant* of block



**Fig. 2.4** List scheduling (left) versus sequence partitioning (right).

distribution, i.e., the maximum number of processors sharing one row or one column of  $A$ .

**Definition 2.10.** Let  $P$  be a partition of  $I \times J$  and let  $P_q$  denote the blocks in  $P$  assigned to processor  $q$ . We define the **sharing constant**  $c_{\text{sh}}$  of  $P$  as

$$c_{\text{sh}} = \max_{i \in I, j \in J} \{c_{\text{sh}}^r(i), c_{\text{sh}}^c(j)\}, \quad (2.4)$$

where  $c_{\text{sh}}^r(i) := |\{q \mid \exists t \times s \in P_q : i \in t\}|$  for  $i \in I$  and  $c_{\text{sh}}^c(j) := |\{q \mid \exists t \times s \in P_q : j \in s\}|$  for  $j \in J$ .

The constant  $c_{\text{sh}}$  can be used to express the costs for sending  $x$  in the first step and for summing up all local vectors  $y_q$  in the last step of Algorithm 2.1. The following definition allows to describe the costs for receiving the vector  $x$  and sending the local result  $y_q$ .

**Definition 2.11.** Let  $P$  be a partition of  $I \times J$  and let  $P_q$  denote the blocks in  $P$  assigned to processor  $q$ . We define

$$I(q) = \bigcup_{t \times s \in P_q} t \quad \text{and} \quad J(q) = \bigcup_{t \times s \in P_q} s$$

as the rows and the columns associated with processor  $q$ . Furthermore, let

$$\rho = \max_{0 \leq q < p} \{|I(q)|, |J(q)|\}.$$

For the above uniform block distribution  $c_{\text{sh}}$  equals  $\sqrt{p}$ , whereas the random distribution induced by list scheduling results in a constant  $c_{\text{sh}}$  which is of order  $p$ . Similarly, we find  $\rho = \max\{|I|, |J|\}/\sqrt{p}$  in the case of a uniform partition and  $\rho = \mathcal{O}(\max\{|I|, |J|\})$  for the random distribution resulting from list scheduling. Using  $c_{\text{sh}}$  and  $\rho$ , (2.3) can be rewritten to obtain the following complexity of the matrix-vector multiplication for general block distributions

$$\mathcal{O}\left(\frac{N_{\text{MV}}(A)}{p} + c_{\text{sh}} \frac{|I| + |J|}{p}\right) + g \cdot \mathcal{O}\left(c_{\text{sh}} \frac{|I| + |J|}{p} + \rho\right) + 3 \cdot \ell. \quad (2.5)$$



Due to the definition of  $\mathcal{H}$ -matrices, i.e., due to the admissibility condition, large blocks tend to be of the order  $I \times J$ . Although exactly this leads to efficient algorithms, it also restricts the possibility of reducing  $\rho$ . Hence, without splitting large matrix blocks, one always ends up with  $\rho = \mathcal{O}(\max\{|I|, |J|\})$ .

Fortunately, this negative result does not apply to  $c_{\text{sh}}$ , which can be reduced using **space-filling curves** and **sequence partitioning**. These methods produce a distribution of  $P$  with a much higher locality of the blocks associated to a specific processor  $q$ ; see Fig. 2.4 (right). Due to the “compactness” of the sets  $P_q$ , the frequency of sharing an index with another processor is reduced.

### 2.3.2.1 Load Balancing with Sequence Partitioning

In this section it will be described how to distribute the blocks among the processors such that on one hand the numerical work for the processors are almost equal and on the other hand  $c_{\text{sh}}$  and  $\rho$  are small. In order to be able to balance the work, we first have to know the costs associated with a processor. Depending on the representations of dense and low-rank matrix blocks, the costs of each block are

$$c_{\text{MV},k}(t,s) = \begin{cases} |t| \cdot |s|, & \text{if } t \times s \in P \text{ is non-admissible,} \\ k(|t| + |s|), & \text{if } t \times s \in P \text{ is admissible.} \end{cases} \quad (2.6)$$

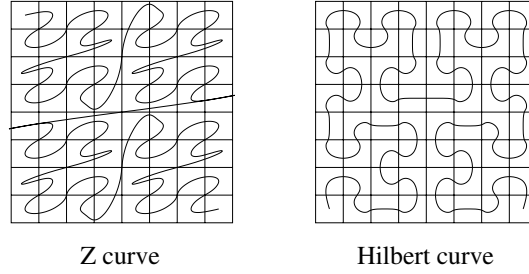
Assume that the set of blocks  $P$  has been rearranged as a sequence. A block distribution will be generated by subdividing this sequence into  $p$  pieces of comparable costs.

**Definition 2.12.** Let  $C = \{c_1, c_2, \dots, c_n\}$  be a sequence of costs  $c_i > 0$ . Furthermore, let  $R = \{r_0, \dots, r_p\}$  with  $1 = r_0 \leq r_1 \leq \dots \leq r_p = n + 1$ ,  $r_i \in \mathbb{N}$ ,  $0 < i < p$ . Then  $R$  is called a **sequence partition** of  $(C, p)$ .  $R$  is **optimal** with respect to  $(C, p)$  if for all partitions  $R' = \{r'_0, \dots, r'_p\}$  of  $C$  it holds that

$$\max_{0 \leq i < p} \sum_{j=r'_i}^{r'_{i+1}-1} c_j \geq \max_{0 \leq i < p} \sum_{j=r_i}^{r_{i+1}-1} c_j =: c_{\max}(C).$$

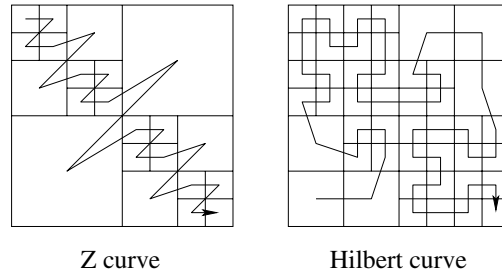
For the computation of an optimal partition of a sequence  $C$ , the knowledge of  $c_{\max}(C)$ , the costs of the most expensive interval in an optimal partition, is sufficient. In [195] an algorithm is presented which computes  $c_{\max}(C)$  with complexity  $\mathcal{O}(n \cdot p)$ . An optimal partition can then be obtained by summing up the costs of each element of the list and starting a new subsequence whenever the costs exceed  $c_{\max}(C)$ .

The required sequence of the blocks in  $P$  can be generated using space filling curves. These curves describe a surjective mapping from the unit interval  $[0, 1]$  to the unit square  $[0, 1]^2$ . Two examples of such curves, the **Z-** and the **Hilbert-curve**, are presented in Fig. 2.5. Since partitions of  $I \times J$  can be mapped to the unit square, the order in which a leaf is reached by the curve defines a sequence usable for sequence partitioning. The neighborhood relationship of adjacent subintervals



**Fig. 2.5** Space-filling curves.

of space-filling curves guarantees the “compactness” of the corresponding sets  $P_q$ . The application of the Z- and the Hilbert-curve to a block partition is depicted in Fig. 2.6.

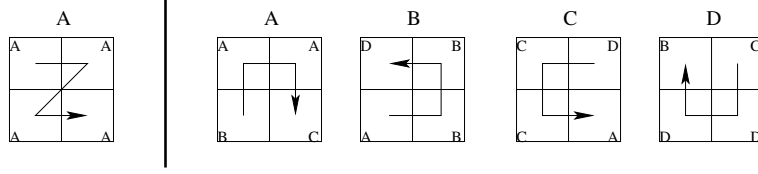


**Fig. 2.6** Space-filling curves applied to  $\mathcal{H}$ -matrices.

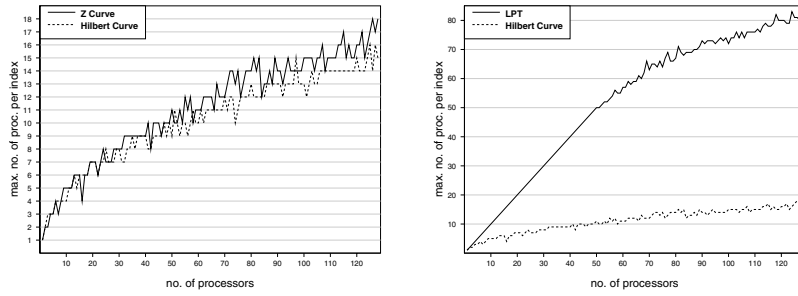
The restriction to quadrees in the definition of block cluster trees allows a simple computation of the ordering induced by space-filling curves. The basic algorithm is a depth first search (DFS) (see [249]) in  $T_{I \times J}$ . In contrast to the usual DFS algorithm, the order in which the sons  $S(b)$  of a node  $b \in T_{I \times J}$  are accessed is defined by a *mark* associated with each node. The marks and the corresponding order of the sons for the Z- and the Hilbert-curve is presented in Fig. 2.7. Here, the root of the block cluster tree always has the mark “A”.

The motivation of load balancing with sequence partitioning was the reduction of the sharing constant  $c_{\text{sh}}$  compared with a random distribution generated by list scheduling. The value of  $c_{\text{sh}}$  obtained using the Z- and the Hilbert-curve for different numbers of processors is shown in Fig. 2.8 (left). For both space-filling curves one can observe a behavior of the kind  $c_{\text{sh}} \sim \sqrt{p}$ , which is equal to the uniform distribution in the case of dense matrices. This shows the reduction of  $c_{\text{sh}}$  in comparison to a random distribution.

We compare the proposed distribution of blocks with another standard scheduling method which is not based on space-filling curves. Instead of assigning the blocks



**Fig. 2.7** Construction of space-filling curves: Z (left) and Hilbert (right).



**Fig. 2.8** Value of  $c_{sh}$  for space-filling curves and LPT scheduling.

randomly to the processors, **longest process time (LPT) scheduling** (cf. [113]) orders the blocks according to their costs, which usually results in a better load balancing than list scheduling. However, it does not reduce  $c_{sh}$ ; see Fig. 2.8 (right). An  $\mathcal{O}(p)$  dependence of  $c_{sh}$  is visible especially for small  $p$ . The number of blocks per processor becomes smaller if  $p > 50$ . Therefore, less processors share the same index.

### 2.3.2.2 Shared Memory Systems

Although communication costs can be neglected on shared memory systems, i.e., it can safely be assumed that  $\ell = g = 0$ , we can use the same algorithm as in the case of a distributed memory machine. This can be justified by examining the (hidden) constants in the part of (2.5) which describes the computational work. Assuming  $c_{sh} \sim \sqrt{p}$ , we can rewrite this equation as

$$\mathcal{O}\left(\frac{N_{MV}(A)}{p} + c \frac{|I| + |J|}{\sqrt{p}}\right)$$

with a small constant  $c > 0$ . On shared memory systems usual values for  $p$  range from 1 to 128. Since the influence of the second term can be seen only for large  $p$ , the first term dominates the computational work. Hence, a high parallel efficiency can also be expected on shared memory systems.

Algorithm 2.1 can be simplified using a threadpool (cf. [165]) based on POSIX threads. For this, the first two steps of the BSP algorithm, i.e., scaling  $y$  and the matrix-vector multiplication, are combined because the vector  $x$  can be accessed by all processors. The summation of the final result is done in a second step after all threads have computed the corresponding local results  $y'_i$ . Another advantage of this algorithm is that for the implementation only minor modifications of an existing sequential version are necessary, e.g., the computation of the matrix-vector product in the first step differs only by the involved set of matrix blocks.

---

```

procedure step_1( $q, \beta, y_q, A_q, x$ )
     $y_q := \beta \cdot y_q$ ;
     $y'_q := \alpha A_q x$ ;
end;

procedure step_2( $q, y_q$ )
     $Y_q := \{y'_j \mid I(j) \cap I(q) \neq \emptyset\}$ ;
     $y_q := y_q + \sum_{y'_j \in Y_q} y'_j$ ;
end;

procedure tp_mv_mul( $\alpha, A, x, \beta, y$ )
    for  $0 \leq q < p$  do run( step_1( $q, \beta, y_q, A_q, x$ ) );
    sync_all();
    for  $0 \leq q < p$  do run( step_2( $q, y_q$ ) );
    sync_all();
end;

```

---

Algorithm 2.2: Matrix-vector multiplication using threads.

### 2.3.3 Numerical Experiments

In this section we examine the performance of the presented parallel matrix-vector multiplication. For simplicity the factors  $\alpha$  and  $\beta$  in the product  $y := \alpha Ax + \beta y$  are chosen 1. In all examples the time for 100 matrix-vector multiplications was measured. We apply the proposed methods to  $\mathcal{H}$ -matrices stemming from the Galerkin discretization of the integral operator from Example 3.43.

*Remark 2.13.* Parallelizing the matrix-vector product cannot be regarded independently of other operations such as generating the matrix approximant. Computing  $\mathcal{H}$ -matrix approximations is usually much more time-consuming than multiplying the approximant by a vector. Therefore, an algorithm for the approximation of discrete integral operators together with its parallelization is presented in Sect. 3.4. Since we should not reassign the blocks to the processors after they have been generated, we will use the block distribution of the matrix-vector multiplication when approximating the matrix. Note that the blocks can be approximated independently while for the matrix-vector multiplication the “compactness” is a critical issue. Here, the problem arises that the rank  $k$  in (2.6) is not known before the matrix

has been generated if a required accuracy has to be satisfied. In this case, we replace  $k$  in (2.6) by a constant  $k_{\text{avg}} = 10$ .

### 2.3.3.1 Shared Memory Systems

For the experiments on a shared memory system an HP9000, PA-RISC with 875 MHz was used. The first comparisons were done employing a square  $\mathcal{H}$ -matrix approximant having a fixed rank of  $k = 10$  on each admissible block. The CPU times resulting from using Algorithm 2.2 and the corresponding parallel efficiency are presented in Table 2.1. The weak parallel performance for small problem sizes  $|I|$

**Table 2.1** 100 parallel matrix-vector multiplications for fixed  $k = 10$ .

$ I $	$p = 1$		$p = 4$		$p = 8$		$p = 12$		$p = 16$	
	time	$E$	time	$E$	time	$E$	time	$E$	time	$E$
3 968	11.7s	88%	3.3s	88%	1.8s	80%	1.3s	73%	1.3s	57%
7 920	30.9s	91%	8.5s	91%	4.6s	84%	3.5s	74%	2.9s	66%
19 320	94.9s	92%	25.9s	92%	13.5s	88%	9.5s	83%	7.5s	79%
43 680	251.7s	89%	70.9s	89%	36.0s	87%	23.9s	88%	18.9s	84%
89 400	556.4s	91%	152.2s	91%	80.0s	87%	53.4s	87%	41.1s	85%
184 040	1277.5s	92%	347.7s	92%	186.0s	86%	120.1s	89%	97.5s	82%

is probably due to the sequential parts in the algorithm, i.e., management overhead. Since this part remains constant independently of the problem size, the parallel efficiency grows with  $|I|$  and stabilizes at about 80–90%.

Table 2.2 shows the results for the same operation but with an  $\mathcal{H}$ -matrix obtained from approximation with fixed accuracy  $\varepsilon = 1_{10}-4$  but variable rank  $k$ . The same

**Table 2.2** 100 parallel matrix-vector multiplications for variable  $k$ .

$ I $	$p = 1$		$p = 4$		$p = 8$		$p = 12$		$p = 16$	
	time	$E$	time	$E$	time	$E$	time	$E$	time	$E$
3 968	9.6s	93%	2.6s	93%	1.6s	73%	1.3s	61%	1.3s	48%
7 920	23.8s	95%	6.3s	95%	3.7s	80%	3.1s	65%	2.4s	63%
19 320	66.7s	97%	17.2s	97%	9.6s	87%	7.0s	80%	5.6s	74%
43 680	169.6s	95%	44.6s	95%	22.8s	93%	15.7s	90%	13.0s	82%
89 400	346.2s	95%	91.1s	95%	47.1s	92%	32.8s	88%	25.7s	84%
184 040	780.5s	96%	202.6s	96%	107.1s	91%	69.8s	93%	55.0s	89%

behavior as in the previous table is visible: the parallel efficiency grows with  $|I|$  and reaches an almost optimal value of about 90%.

### 2.3.3.2 Distributed Memory Systems

The following tests were carried out on an AMD Athlon 900 MHz cluster. Due to memory restrictions, problems for large  $|I|$  could not be computed with a small number of processors  $p$ . The corresponding parallel efficiency for these problem sizes is therefore computed with respect to the smallest available  $p$ , i.e.,

$$E(p) := \frac{p' \cdot t(p')}{p \cdot t(p)},$$

where  $p'$  denotes the smallest number of processors which was able to compute the problem. The presented storage size in these cases is approximated by  $p'N_{\text{st}}$ , where  $N_{\text{st}}$  denotes the memory consumption per processor on a system with  $p'$  CPUs. The results from Table 2.3 were obtained for fixed rank  $k = 10$ . One observes the same

**Table 2.3** 100 parallel matrix-vector multiplications for fixed  $k = 10$ .

$ I $	$p = 1$		$p = 4$		$p = 8$		$p = 12$		$p = 16$	
	time	$E$	time	$E$	time	$E$	time	$E$	time	$E$
3 968	16.2s		4.8s	85%	2.8s	72%	2.1s	65%	1.6s	65%
7 920	43.8s		12.1s	91%	6.7s	81%	4.8s	76%	4.0s	69%
19 320	141.1s		39.5s	89%	20.2s	87%	14.7s	80%	11.1s	80%
43 680			107.9s		57.0s	95%	42.0s	86%	32.1s	84%
89 400					129.9s		90.8s	95%	69.7s	93%
184 040							209.4s		157.4s	100%

behavior for the parallel performance as in the case of a shared memory system: a better efficiency is obtained for larger  $|I|$ . This effect is also visible for fixed accuracy  $\varepsilon = 10^{-4}$  as the results in Table 2.4 indicate. Due to the approximation of the actual

**Table 2.4** 100 parallel matrix-vector multiplications for variable  $k$ .

$ I $	$p = 1$		$p = 4$		$p = 8$		$p = 12$		$p = 16$	
	time	$E$	time	$E$	time	$E$	time	$E$	time	$E$
3 968	12.0s		3.6s	83%	2.1s	71%	2.0s	50%	1.3s	57%
7 920	30.0s		8.6s	87%	5.0s	76%	3.6s	69%	3.0s	64%
19 320	84.6s		27.0s	79%	13.2s	80%	9.5s	74%	7.5s	70%
43 680	221.0s		64.0s	86%	34.5s	80%	23.2s	80%	25.5s	54%
89 400					74.1s		53.6s	92%	42.4s	87%
184 040							119.6s		90.8s	99%

costs (see Remark 2.13), the parallel efficiency is not as high as for an  $\mathcal{H}$ -matrix with fixed rank.

As a conclusion of these test, we observe that starting from an existing sequential implementation of  $\mathcal{H}$ -matrices, only a minimal programming effort is necessary to

make use of multiple processors on a shared memory machine. The resulting algorithms show a high parallel efficiency and are therefore recommended for the acceleration of the  $\mathcal{H}$ -matrix arithmetic on workstations and compute-servers. If a larger number of processors is needed, distributed memory machines are usually preferred due to their lower costs. Using the BSP model, the design and implementation of parallel algorithms on such computer systems is similar to shared memory systems. The corresponding parallel matrix-vector multiplication also shows a high parallel efficiency if the problem size is sufficiently large.

While the matrix-vector multiplication is exact up to machine precision, the following replacements of the usual matrix operations are approximate. Since most of the algorithms guarantee a prescribed accuracy on each block, it is important to be able to relate blockwise accuracy estimates to global ones.

## 2.4 Blockwise and Global Norms

From the analysis we will usually obtain estimates on each of the blocks  $b$  of a partition  $P$ . However, such estimates are finally required for the whole matrix. If we are interested in the Frobenius norm, blockwise estimates directly translate to global estimates:

$$\|A_b\|_F \leq \varepsilon \text{ for all } b \in P \implies \|A\|_F \leq \sqrt{|P|} \varepsilon$$

and

$$\|A_b\|_F \leq \|B_b\|_F \text{ for all } b \in P \implies \|A\|_F \leq \|B\|_F.$$

Both implications follow from

$$\|A\|_F^2 = \sum_{b \in P} \|A_b\|_F^2. \quad (2.7)$$

For the spectral norm the situation is a bit more difficult. We can, however, exploit the structure of the partition  $P$  together with the following lemma.

**Lemma 2.14.** *Consider the following  $r \times r$  block matrix*

$$A = \begin{bmatrix} A_{11} & \dots & A_{1r} \\ \vdots & & \vdots \\ A_{r1} & \dots & A_{rr} \end{bmatrix} \quad (2.8)$$

with  $A_{ij} \in \mathbb{C}^{m_i \times n_j}$ ,  $i, j = 1, \dots, r$ . Then it holds that

$$\max_{i,j=1,\dots,r} \|A_{ij}\|_2 \leq \|A\|_2 \leq \left( \max_{i=1,\dots,r} \sum_{j=1}^r \|A_{ij}\|_2 \right)^{1/2} \left( \max_{j=1,\dots,r} \sum_{i=1}^r \|A_{ij}\|_2 \right)^{1/2}. \quad (2.9)$$

*Proof.* Let  $x = [x_1, \dots, x_r]^T \in \mathbb{C}^n$ , where  $x_j \in \mathbb{C}^{n_j}$ ,  $j = 1, \dots, r$ , and  $n := \sum_{j=1}^r n_j$ . Observe that

$$\|Ax\|_2^2 = \sum_{i=1}^r \left\| \sum_{j=1}^r A_{ij}x_j \right\|_2^2 \leq \sum_{i=1}^r \left( \sum_{j=1}^r \|A_{ij}\|_2 \|x_j\|_2 \right)^2 = \|\hat{A}\hat{x}\|_2^2,$$

where  $\hat{A} \in \mathbb{R}^{r \times r}$  has the entries  $\hat{a}_{ij} = \|A_{ij}\|_2$  and  $\hat{x} \in \mathbb{R}^r$  is the vector with components  $\hat{x}_j = \|x_j\|_2$ ,  $j = 1, \dots, r$ . It is well known that  $\|\hat{A}\|_2^2 \leq \|\hat{A}\|_1 \|\hat{A}\|_\infty$ . Hence,

$$\|\hat{A}\hat{x}\|_2^2 \leq \|\hat{A}\|_1 \|\hat{A}\|_\infty \|\hat{x}\|_2^2 = \|\hat{A}\|_1 \|\hat{A}\|_\infty \|x\|_2^2$$

gives the first part of the assertion. The lower bound follows from the fact that the spectral norm of any sub-block of a matrix  $A$  is bounded by the spectral norm of  $A$ .  $\square$

For block matrices generated from recursive subdivision we obtain

**Theorem 2.15.** *Assume that the partition  $P$  is generated from  $I \times J$  by recursively subdividing each block into a  $2 \times 2$  block structure at most  $L$  times. Furthermore, let  $A, B \in \mathbb{C}^{I \times J}$  such that  $\|A_b\|_2 \leq \|B_b\|_2$  for all blocks  $b \in P$ . Then it holds that*

$$\|A\|_2 \leq 2^L \|B\|_2.$$

*Proof.* The assertion is proved by induction over the depth  $L$  of the cluster tree. The estimate is trivial if  $L = 0$ . Assume that the assertion holds for an  $L \in \mathbb{N}$ . Let

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

have depth  $L + 1$ . Since  $A_{ij}$ ,  $i, j = 1, 2$ , have depth  $L$ , we know from the induction that

$$\|A_{ij}\|_2 \leq 2^L \|B_{ij}\|_2, \quad i, j = 1, 2.$$

The previous lemma shows

$$\|A\|_2 = \left\| \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \right\|_2 \leq 2 \max_{i,j=1,2} \|A_{ij}\|_2 \leq 2^{L+1} \max_{i,j=1,2} \|B_{ij}\|_2 \leq 2^{L+1} \|B\|_2,$$

which proves the assertion.  $\square$

For the usual choice  $L \sim \min\{\log_2 |I|, \log_2 |J|\}$ , the coefficient  $2^L$  in the previous theorem will be of the order  $\min\{|I|, |J|\}$ . If more structure of  $P$  than just a recursive subdivision is known, then this estimate can be significantly improved. Note that block cluster trees have  $|T_{I \times J}| \sim \min\{|I|, |J|\}$  elements while recursive subdivision in general leads to  $|I| \cdot |J|$  blocks.

An important consequence of (2.9) is that for matrices (2.8) vanishing in all but  $\nu$  blocks in each row and each column it follows that

$$\max_{i,j=1,\dots,r} \|A_{ij}\|_2 \leq \|A\|_2 \leq \nu \max_{i,j=1,\dots,r} \|A_{ij}\|_2.$$



The previous estimate was also proved in [114] with a different technique. This equivalence of the global and the blockwise spectral norm is useful in translating blockwise errors to a global one. When relative error estimates are to be derived, we will additionally need to estimate how a local norm relation is carried over to the whole matrix.

**Theorem 2.16.** *Let  $P$  be the leaves of a block cluster tree  $T_{I \times J}$ . Then for  $A, B \in \mathcal{H}(T_{I \times J}, k)$  it holds that*

- (i)  $\max_{b \in P} \|A_b\|_2 \leq \|A\|_2 \leq c_{\text{sp}} L(T_{I \times J}) \max_{b \in P} \|A_b\|_2$ ;
- (ii)  $\|A\|_2 \leq c_{\text{sp}} L(T_{I \times J}) \|B\|_2$  provided  $\max_{b \in P} \|A_b\|_2 \leq \max_{b \in P} \|B_b\|_2$ .

*Proof.* Let  $A_\ell$  denote the part of  $A$  which corresponds to the blocks of  $P$  from the  $\ell$ th level of  $T_{I \times J}$ ; i.e.,

$$(A_\ell)_b = \begin{cases} A_b, & b \in T_{I \times J}^{(\ell)} \cap P, \\ 0, & \text{else.} \end{cases}$$

Then  $A = \sum_{\ell=1}^{L(T_{I \times J})} A_{\ell-1}$ . Since  $A_\ell$  has tensor structure with at most  $c_{\text{sp}}$  blocks per block row or block column, Lemma 2.14 gives  $\|A_\ell\|_2 \leq c_{\text{sp}} \max_{b \in T_{I \times J}^{(\ell)} \cap P} \|A_b\|_2$ , such that

$$\|A\|_2 \leq \sum_{\ell=1}^{L(T_{I \times J})} \|A_{\ell-1}\|_2 \leq c_{\text{sp}} \sum_{\ell=1}^{L(T_{I \times J})} \max_{b \in T_{I \times J}^{(\ell-1)} \cap P} \|A_b\|_2 \leq c_{\text{sp}} L(T_{I \times J}) \max_{b \in P} \|A_b\|_2.$$

The estimate

$$\max_{b \in P} \|A_b\|_2 \leq \max_{b \in P} \|B_b\|_2 \leq \|B\|_2$$

gives the second part of the assertion.  $\square$

The computation of the Frobenius norm of  $A \in \mathcal{H}(T_{I \times J}, k)$  can be done using (2.7) and (1.4) with at most  $c_{\text{sp}} \max\{k^2, n_{\min}\} [|I| \log |I| + |J| \log |J|]$  arithmetic operations. The spectral norm of  $A$  can also be computed with logarithmic-linear complexity, for instance, by the power method applied to  $A^H A$ , which  $A$  enters only through the matrix-vector product.

## 2.5 Adding $\mathcal{H}$ -Matrices

The sum of two matrices  $A, B \in \mathcal{H}(T_{I \times J}, k)$  will usually be in  $\mathcal{H}(T_{I \times J}, 2k)$  but not in  $\mathcal{H}(T_{I \times J}, k)$ . The reason for this is that  $\mathbb{C}_k^{m \times n}$  is not a linear space as we have seen in Sect. 1.1.5. Hence,  $\mathcal{H}(T_{I \times J}, k)$  is not a linear space and we have to approximate the sum  $A + B$  by a matrix  $S \in \mathcal{H}(T_{I \times J}, k)$  if we want to avoid that the rank and hence the complexity grows with each addition. Obviously, this can be done using the rounded addition from Sect. 1.1.5 on each admissible block. On non-admissible block, the usual (entrywise) addition is employed. The addition of  $\mathcal{H}$ -matrices can therefore easily be parallelized using the scheduling algorithms from Sect. 2.3.2.1.

Since the rounded addition gives a blockwise best approximation (see Theorem 1.7),  $S$  is a best approximation in the Frobenius norm

$$\|A + B - S\|_F \leq \|A + B - M\|_F \quad \text{for all } M \in \mathcal{H}(T_{I \times J}, k).$$

Using Theorem 2.16, this estimate for the spectral norm reads

$$\|A + B - S\|_2 \leq c_{\text{sp}} L(T_{I \times J}) \|A + B - M\|_2 \quad \text{for all } M \in \mathcal{H}(T_{I \times J}, k).$$

The following bound on the number of arithmetic operations results from Theorem 1.7, (1.35), and (1.36).

**Theorem 2.17.** *Let  $A, B \in \mathcal{H}(T_{I \times J}, k)$ . The number of operations required for computing a matrix  $S \in \mathcal{H}(T_{I \times J}, k)$  satisfying the above error estimates is of the order*

$$c_{\text{sp}} k^2 [L(T_I)|I| + L(T_J)|J|] + c_{\text{sp}} k^3 \min\{|T_I|, |T_J|\}.$$

Alternatively, not  $k$  but the blockwise accuracy  $\varepsilon$  of the approximation can be prescribed; i.e.,

$$\|(A + B)_b - S_b\|_2 \leq \varepsilon \|(A + B)_b\|_2 \quad \text{for all } b \in P.$$

In this case, the required blockwise rank depends on the matrices and cannot be predicted without deeper knowledge of the underlying problem. Using Theorem 2.16, we obtain the relative error estimate

$$\|A + B - S\|_2 \leq c_{\text{sp}} L(T_{I \times J}) \varepsilon \|A + B\|_2.$$

### 2.5.1 Preserving Positivity

In the rest of this chapter we will also define replacements for other common matrix operations. Since these substitutes will all be based on the rounded addition, the introduced error will propagate and will in particular perturb the eigenvalues of the results of these operations. If the smallest eigenvalue is close to the origin compared with the rounding accuracy  $\varepsilon$ , it may happen that the result of these operations becomes indefinite although it should be positive definite in exact arithmetic. Such a situation can be avoided by the following ideas; cf. [28].

Assume that  $\hat{A} \in \mathbb{C}^{I \times I}$  is the Hermitian positive definite result of an exact addition of two matrices from  $\mathcal{H}(T_{I \times I}, k)$  and let  $A \in \mathcal{H}(T_{I \times I}, k)$  be its  $\mathcal{H}$ -matrix approximant. For a moment we assume that  $\hat{A}$  and  $A$  differ only on a single off-diagonal block  $t \times s \in P$ . Let  $EF^H$ ,  $E \in \mathbb{C}^{t \times k}$ ,  $F \in \mathbb{C}^{s \times k}$ , be the error matrix associated with  $t \times s$ ; i.e.,

$$A_{ts} = \hat{A}_{ts} - EF^H$$

and let

$$\varepsilon := \max\{\|E\|_2^2, \|F\|_2^2\}. \quad (2.10)$$

Due to symmetry,  $FE^H$  is the error matrix on block  $s \times t$ .

We modify the approximant  $A$  in such a manner that the new approximant  $\tilde{A}$  can be guaranteed to be positive definite. This is done by adding  $EE^H$  to  $A_{tt}$  and  $FF^H$  to  $A_{ss}$  such that

$$\begin{bmatrix} \tilde{A}_{tt} & \tilde{A}_{ts} \\ \tilde{A}_{ts}^H & \tilde{A}_{ss} \end{bmatrix} := \begin{bmatrix} A_{tt} & A_{ts} \\ A_{ts}^H & A_{ss} \end{bmatrix} + \begin{bmatrix} EE^H & \\ & FF^H \end{bmatrix} = \begin{bmatrix} \hat{A}_{tt} & \hat{A}_{ts} \\ \hat{A}_{ts}^H & \hat{A}_{ss} \end{bmatrix} + \begin{bmatrix} EE^H & -EF^H \\ -FE^H & FF^H \end{bmatrix}.$$

Since

$$\begin{bmatrix} EE^H & -EF^H \\ -FE^H & FF^H \end{bmatrix} = \begin{bmatrix} -E \\ F \end{bmatrix} \begin{bmatrix} -E \\ F \end{bmatrix}^H$$

is positive semi-definite, the eigenvalues of  $\tilde{A}$  are not smaller than those of  $\hat{A}$ . Therefore,  $\tilde{A}$  is Hermitian positive definite and

$$\left\| \begin{bmatrix} \tilde{A}_{tt} & \tilde{A}_{ts} \\ \tilde{A}_{ts}^H & \tilde{A}_{ss} \end{bmatrix} - \begin{bmatrix} \hat{A}_{tt} & \hat{A}_{ts} \\ \hat{A}_{ts}^H & \hat{A}_{ss} \end{bmatrix} \right\|_2 = \left\| \begin{bmatrix} EE^H & -EF^H \\ -FE^H & FF^H \end{bmatrix} \right\|_2 \leq \|E\|_2^2 + \|F\|_2^2 \leq 2\varepsilon.$$

If a relative error is preferred, we have to guarantee that

$$\|E\|_2^2 \leq \varepsilon \|\hat{A}_{tt}\|_2, \quad \|F\|_2^2 \leq \varepsilon \|\hat{A}_{ss}\|_2 \quad \text{and} \quad \|EF^H\|_2 \leq \varepsilon \|\hat{A}_{ts}\|_2$$

holds instead of (2.10). In this case, we obtain from Theorem 2.16 that

$$\left\| \begin{bmatrix} \tilde{A}_{tt} & \tilde{A}_{ts} \\ \tilde{A}_{ts}^H & \tilde{A}_{ss} \end{bmatrix} - \begin{bmatrix} \hat{A}_{tt} & \hat{A}_{ts} \\ \hat{A}_{ts}^H & \hat{A}_{ss} \end{bmatrix} \right\|_2 = \left\| \begin{bmatrix} EE^H & -EF^H \\ -FE^H & FF^H \end{bmatrix} \right\|_2 \leq 2\varepsilon \left\| \begin{bmatrix} \hat{A}_{tt} & \hat{A}_{ts} \\ \hat{A}_{ts}^H & \hat{A}_{ss} \end{bmatrix} \right\|_2.$$

Since  $t \times t$  and  $s \times s$  will usually not be leaves in  $T_{I \times I}$ , it is necessary that  $EE^H$  and  $FF^H$  are restricted to the leaves of  $t \times t$  and  $s \times s$  when adding them to  $A_{tt}$  and  $A_{ss}$ , respectively. Note that this leads to a rounding error which in turn has to be added to the diagonal sub-blocks of  $t \times t$  and  $s \times s$  in order to preserve positivity. The computational complexity which is connected with the rounded addition makes it necessary to improve the above idea. Once again, we replace an approximant with another approximant by adding a positive semi-definite matrix. Let  $t_1$  and  $t_2$  be the sons of  $t$  and let  $s_1$  and  $s_2$  be the sons of  $s$ . If we define

$$\tilde{\tilde{A}}_{tt} := \tilde{A}_{tt} + \begin{bmatrix} -E_{t_1} \\ E_{t_2} \end{bmatrix} \begin{bmatrix} -E_{t_1} \\ E_{t_2} \end{bmatrix}^H = A_{tt} + 2 \begin{bmatrix} E_{t_1} E_{t_1}^H & 0 \\ 0 & E_{t_2} E_{t_2}^H \end{bmatrix},$$

the problem of adding  $EE^H$  to  $A_{tt}$  is reduced to adding  $2E_{t_1}E_{t_1}^H$  to  $A_{t_1t_1}$  and  $2E_{t_2}E_{t_2}^H$  to  $A_{t_2t_2}$ . Applying this idea recursively, adding  $EE^H$  to  $A_{tt}$  can finally be done by adding a multiple of  $E_{t'}E_{t'}^H$  to the dense matrix block  $A_{t't'}$  for each leaf  $t'$  in  $T_t$  from the set of descendants of  $t$ . We obtain the following two algorithms `addsym_stab` and `addsym_diag`.

---

```

procedure addsym_stab( $t, s, U, V, \text{var } A$ )
if  $t \times s$  is non-admissible then
    add  $UV^H$  to  $A_{ts}$  without approximation;
else
    add  $UV^H$  to  $A_{ts}$  using the rounded addition;
    denote by  $EF^H$  the rounding error;
    addsym_diag( $t, E, A$ );
    addsym_diag( $s, F, A$ );
endif

```

---

Algorithm 2.3: Stabilized Hermitian rounded addition.

The first adds a matrix of low rank  $UV^H$  to an off-diagonal block  $t \times s$  while the latter adds  $EE^H$  to the diagonal block  $t \times t$ . Note that we assume that an Hermitian matrix is represented by its upper triangular part only.

---

```

procedure addsym_diag( $t, E, \text{var } A$ )
if  $t \times t$  is a leaf then
    add  $EE^H$  to  $A_{tt}$  without approximation;
else
    addsym_diag( $t_1, \sqrt{2}E_{t_1}, A$ );
    addsym_diag( $t_2, \sqrt{2}E_{t_2}, A$ );
endif

```

---

Algorithm 2.4: Stabilized diagonal addition.

We will now estimate the costs if the above algorithms are applied to  $t \times s \in T_{I \times I}$ . Denote by  $N_{\text{diag}}^{\text{stab}}(t)$  the number of operations needed if Algorithm 2.4 is applied to  $t \in T_I \setminus \mathcal{L}(T_I)$  with  $E \in \mathbb{C}^{t \times k}$ . Since

$$N_{\text{diag}}^{\text{stab}}(t) = N_{\text{diag}}^{\text{stab}}(t_1) + N_{\text{diag}}^{\text{stab}}(t_2)$$

and since at most  $k|t'|^2$  operations are required on each leaf  $t'$  of  $T_t$ , we obtain

$$N_{\text{diag}}^{\text{stab}}(t) = \sum_{t' \in \mathcal{L}(T_t)} N_{\text{diag}}^{\text{stab}}(t') \leq \sum_{t' \in \mathcal{L}(T_t)} k|t'|^2 \leq n_{\min} k \sum_{t' \in \mathcal{L}(T_t)} |t'| = n_{\min} k |t|.$$

Additionally, denote by  $N_{\text{add}}^{\text{stab}}(t, s)$  the number of operations required to add  $UV^H \in \mathbb{C}_k^{t \times s}$  to  $A_{ts}$  using Algorithm 2.3. If  $t \times s$  is non-admissible, then  $\min\{|t|, |s|\} \leq n_{\min}$ , which leads to

$$N_{\text{add}}^{\text{stab}}(t, s) \leq |t||s| \leq n_{\min}(|t| + |s|).$$

Since for admissible  $t \times s \in T_{I \times I}$  a rounded addition and two calls to `addsym_diag` have to be performed, the costs of Algorithm 2.4 can be estimated by

$$\begin{aligned} N_{\text{add}}^{\text{stab}}(t, s) &= \max\{k^2, n_{\min}\}(|t| + |s|) + N_{\text{diag}}^{\text{stab}}(t) + N_{\text{diag}}^{\text{stab}}(s) \\ &\leq [\max\{k^2, n_{\min}\} + n_{\min}k](|t| + |s|). \end{aligned}$$

Hence, the stabilized addition has asymptotically the same computational complexity as the rounded addition on each block.

If two  $\mathcal{H}$ -matrices are to be added, the stabilized addition has to be applied to each block. The resulting  $\mathcal{H}$ -matrix will differ from the result  $S$  of the approximate addition from Sect. 2.5 only in the diagonal blocks of  $P$ . Hence, it requires the same amount of storage. The following theorem gathers the estimates of this section.

**Theorem 2.18.** *Let  $A, B$  be Hermitian and let  $\lambda_i$ ,  $i \in I$ , denote the eigenvalues of  $A + B$ . Assume that  $S \in \mathcal{H}(T_{I \times I}, k)$  has precision  $\varepsilon$ . Using the stabilized rounded addition on each block leads to a matrix  $\tilde{S} \in \mathcal{H}(T_{I \times I}, k)$  with eigenvalues  $\tilde{\lambda}_i \geq \lambda_i$ ,  $i \in I$ , satisfying*

$$\|A + B - \tilde{S}\|_2 \sim L(T_I)|I|\varepsilon.$$

*Hence, if  $A + B$  is positive definite, so is  $\tilde{S}$ . At most  $(\max\{k^2, n_{\min}\} + n_{\min}k)L(T_I)|I|$  operations are required for the construction of  $\tilde{S}$ .*

*Proof.* Let  $t \in T_I^{(\ell)}$  be a cluster from the  $\ell$ th level of  $T_I$ . Since at most  $c_{\text{sp}}$  blocks  $t \times s$ ,  $s \in T_I$ , are contained in  $P$ , `addsym_diag` is applied to  $t$  only  $c_{\text{sp}}$  times during the stabilized addition of  $A$  and  $B$ . This routine adds terms  $2^p E_{t'} E_{t'}^H$ ,  $p < L(T_I) - \ell$ , to  $S_{t't'}$ . Hence, the error on  $t' \times t'$  is bounded by

$$c_{\text{sp}} \sum_{\ell=0}^{L(T_I)} 2^{L(T_I)-1-\ell} \varepsilon \leq c_{\text{sp}} 2^{L(T_I)} \varepsilon \leq c c_{\text{sp}} |I| \varepsilon$$

with some constant  $c > 0$ . The previous estimate follows from the fact that the depth  $L(T_I)$  of  $T_I$  scales like  $\log_2 |I|$ . Since all other blocks coincide with the blocks of  $S$ , which have accuracy  $\varepsilon$ , we obtain the estimate

$$\|A + B - \tilde{S}\|_2 \leq c c_{\text{sp}}^2 L(T_{I \times I}) |I| \varepsilon \leq c c_{\text{sp}}^2 L(T_I) |I| \varepsilon$$

due to Theorem 2.16.  $\square$

The stabilized addition will be used in Sect. 3.6.3 when computing approximations to Cholesky decompositions of almost singular matrices.

## 2.6 Coarsening $\mathcal{H}$ -Matrices

In this section we describe how a given matrix  $A \in \mathcal{H}(T_{I \times J}, k)$  is approximated by a matrix  $\tilde{A} \in \mathcal{H}(T'_{I \times J}, k')$ , where  $T'_{I \times J}$  is a sub-tree of  $T_{I \times J}$  with the same root  $I \times J$ . In the first part of this section,  $k' \leq k$  will be a given number such that the accuracy of  $\tilde{A}$  can be estimated only relatively to the best approximation. In the second part we prescribe the accuracy and estimate the resulting rank  $k'$ .

We have already got to know the following two coarsening techniques.

- (a) *Blockwise coarsening:* Approximants of lower accuracy compared with the accuracy of  $A$  are, for instance, sufficient when generating preconditioners of  $A$ . In order to improve the data-sparsity and thereby the efficiency of the  $\mathcal{H}$ -matrix

approximant, it is helpful to remove superfluous information from the blocks. In Sect. 1.1.3 it was described how to compute a low-rank approximant of prescribed accuracy and minimal rank to a given low-rank matrix. Especially in the case of non-local operators (see Chap. 3), this recompression technique is likely to improve the storage requirements even if the accuracy is not reduced. The reason for this is that low-rank approximations are usually generated from non-optimal constructions.

- (b) *Agglomeration of blocks*: The partition  $P$  generated in Sect. 1.3 is admissible and can be computed with logarithmic-linear complexity. We have remarked that  $P$ , however, may be non-optimal. Hence, there is a good chance to improve it by agglomerating blocks using the procedure from Sect. 1.1.6; see also [115]. The agglomeration of blocks will be particularly beneficial to the efficiency of arithmetic operations such as multiplication and inversion of  $\mathcal{H}$ -matrices due to an improved sparsity constant.

Coarsening can be applied to a whole  $\mathcal{H}$ -matrix but also to a submatrix. Without loss of generality we consider only the case that  $T'_{I \times J} = I \times J$ ; i.e.,  $A$  is coarsened to a single matrix block  $\tilde{A}$  of rank  $k'$ .

### 2.6.0.1 Coarsening with Prescribed Rank

For the first part of this section assume that  $k' \leq k$  is given. We start from the tree  $T_L := T_{I \times J}$ ,  $L := L(T_{I \times J}) - 1$ , and a matrix  $A_L \in \mathcal{H}(T_L, k')$  which is generated from  $A$  by approximating each block  $A_b$ ,  $b \in \mathcal{L}(T_{I \times J})$ , by a matrix of rank  $k'$  using the technique from Sect. 1.1.3. This first coarsening step requires

$$\sum_{t \times s \in \mathcal{L}(T_{I \times J})} 6k^2(|t| + |s|) + 20k^3 \leq 6c_{\text{sp}}k^2L(T_{I \times J})[|I| + |J|] + 40c_{\text{sp}}k^3 \min\{|I|, |J|\}/n_{\min}$$

arithmetic operations due to (1.36) and Lemma 1.39. Since  $A$  is approximated on each block by a best approximation, for the Frobenius norm it holds that

$$\|A - A_L\|_F \leq \|A - M\|_F \quad \text{for all } M \in \mathcal{H}(T_L, k'). \quad (2.11)$$

The following rule defines a sequence of block cluster trees  $T_\ell$  and an associated sequence of approximants  $A_\ell \in \mathcal{H}(T_\ell, k')$ ,  $\ell = L - 1, \dots, 0$ . Let  $T_\ell$  result from  $T_{\ell+1}$  by removing the sons of each block  $b \in T_{\ell+1}^{(\ell)} \setminus \mathcal{L}(T_{\ell+1})$  in the  $\ell$ th level of  $T_{\ell+1}$ .  $A_\ell$  results from  $A_{\ell+1}$  by the agglomeration procedure from Sect. 1.1.6 applied to such blocks  $b$ . The property that a best approximation is attained on each block is inherited by the whole matrix with respect to the Frobenius norm; i.e.,

$$\|A_{\ell+1} - A_\ell\|_F \leq \|A_{\ell+1} - M\|_F \quad \text{for all } M \in \mathcal{H}(T_\ell, k'). \quad (2.12)$$

In order to agglomerate a non-admissible block, it is first converted to the outer-product representation using the SVD.

The number of arithmetic operations of the above construction is determined by the number of operations required for the SVD of each non-admissible block and the numerical effort of the agglomeration of each block  $b \in T_{I \times J} \setminus \mathcal{L}(T_{I \times J})$ . According to Sect. 1.1.6, at most

$$\sum_{t \times s \in T_{I \times J}} 24k^2(|t| + |s|) + \max\{1408\frac{2}{3}k^3, 22n_{\min}^3\}$$

operations are required, where we have used that  $\max\{|t|, |s|\} \leq n_{\min}$ , which may be assumed due to the same level of a block's row and column cluster. Using (1.35) we obtain

**Lemma 2.19.** *The number of arithmetic operations required for the above construction of  $A_0$  is of the order*

$$c_{\text{sp}}k^2L(T_{I \times J})(|I| + |J|) + \max\{k^3, n_{\min}^3\}|T_{I \times J}|.$$

Using the above procedure,  $A \in \mathcal{H}(T_{I \times J}, k)$  is rounded to a matrix of rank  $k'$ . The resulting approximation error can be arbitrarily bad. Assume we know a best approximant  $A_{\text{best}} \in \mathbb{C}_{k'}^{I \times J}$  for  $A$ . The following lemma (cf. [116]) relates the approximation error  $\|A - A_0\|_F$  to the best possible error  $\|A - A_{\text{best}}\|_F$ .

**Lemma 2.20.** *Let  $A_0 \in \mathbb{C}_{k'}^{I \times J}$  be constructed as above. Then it holds that*

$$\|A - A_0\|_F \leq 2^{L(T_{I \times J})} \|A - A_{\text{best}}\|_F.$$

*Proof.* Let  $L = L(T_{I \times J}) - 1$ . From (2.12) it follows that

$$\|A_\ell - A_{\text{best}}\|_F \leq \|A_\ell - A_{\ell+1}\|_F + \|A_{\ell+1} - A_{\text{best}}\|_F \leq 2\|A_{\ell+1} - A_{\text{best}}\|_F$$

for  $\ell \in \{0, \dots, L-1\}$ , because  $A_{\text{best}} \in \mathcal{H}(T_\ell, k')$ . Hence,  $\|A_\ell - A_{\text{best}}\|_F \leq 2^{L-\ell} \|A_L - A_{\text{best}}\|_F$  and we obtain from (2.12)

$$\begin{aligned} \|A_L - A_0\|_F &= \left\| \sum_{\ell=0}^{L-1} (A_\ell - A_{\ell+1}) \right\|_F \leq \sum_{\ell=0}^{L-1} \|A_\ell - A_{\ell+1}\|_F \leq \sum_{\ell=0}^{L-1} \|A_{\ell+1} - A_{\text{best}}\|_F \\ &\leq \sum_{\ell=0}^{L-1} 2^{L-\ell-1} \|A_L - A_{\text{best}}\|_F = (2^L - 1) \|A_L - A_{\text{best}}\|_F. \end{aligned}$$

The assertion follows from

$$\begin{aligned} \|A - A_0\|_F &\leq \|A - A_L\|_F + \|A_L - A_0\|_F \leq \|A - A_L\|_F + (2^L - 1) \|A_L - A_{\text{best}}\|_F \\ &\leq 2^L \|A - A_L\|_F + (2^L - 1) \|A - A_{\text{best}}\|_F \leq (2^{L+1} - 1) \|A - A_{\text{best}}\|_F \end{aligned}$$

due to (2.11).  $\square$

### 2.6.0.2 Coarsening with Prescribed Accuracy

If on the other hand a given accuracy  $\varepsilon > 0$  is to be satisfied in each agglomeration step, i.e., we have

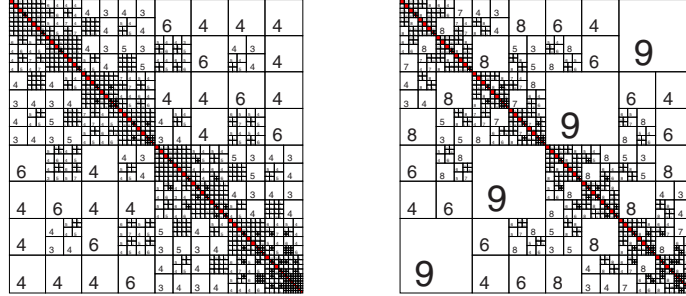
$$\|A_{\ell+1} - A_\ell\|_F \leq \varepsilon \|A_{\ell+1}\|_F, \quad 0 \leq \ell < L, \quad (2.13)$$

then the required rank  $k'$  may increase. In order to avoid that the complexity is deteriorated, we stop the coarsening process in blocks for which the required rank  $k'$  is such that agglomeration is not worthwhile.

Assume that the sub-blocks  $S(b)$  of a block  $b = t \times s \in T_{\ell+1}^{(\ell)} \setminus \mathcal{L}(T_{\ell+1})$  in  $A_{\ell+1}$  are low-rank matrices with ranks  $k_{t' \times s'}, t' \times s' \in S(b)$ . By comparing the original storage costs of  $(A_{\ell+1})_b$  with those of  $(A_\ell)_b$ , it is easy to check whether the coarsening leads to reduced costs of the approximant. If

$$k_{t \times s}(|t| + |s|) \leq \sum_{t' \times s' \in S(b)} k_{t' \times s'}(|t'| + |s'|), \quad (2.14)$$

then the block cluster tree  $T_{\ell+1}$  is modified by replacing the sons  $S(b)$  of  $b$  by the new leaf  $b$ . If this condition is not satisfied, then the sons of  $b$  will be kept in the block cluster tree. This procedure can then be applied to the leaves of the new block cluster tree until (2.14) is not satisfied.



**Fig. 2.9**  $\mathcal{H}$ -matrix before and after coarsening.

It is obvious that this procedure does not increase the amount of storage. In contrast, depending on  $A$  it will usually improve the storage requirements. In the following two lemmas (see [28]) we analyze the accuracy and the complexity of this adaptive agglomeration process. For this purpose we consider a single block  $b \in T_{I \times J}$  in which the described agglomeration stops due to the violation of (2.14) by the father block of  $b$ . Without loss of generality we may identify  $b$  with  $I \times J$  and assume that (2.14) holds for all  $t \times s \in T_{I \times J} \setminus \mathcal{L}(T_{I \times J})$ . The following lemma describes the accuracy of  $A_0$  compared with the accuracy of  $A_L$ .



**Lemma 2.21.** *Let  $A \in \mathbb{C}^{I \times J}$ . If  $\|A - A_L\|_F \leq \varepsilon \|A\|_F$ , then  $\|A - A_0\|_F \leq c(\varepsilon) \|A\|_F$ , where  $c(\varepsilon) = \varepsilon + (1 + \varepsilon)[(1 + \varepsilon)^L - 1] \sim L(T_{I \times J})\varepsilon$  for  $\varepsilon \rightarrow 0$ .*

*Proof.* Due to  $\|A_\ell\|_F \leq \|A_{\ell+1}\|_F + \|A_{\ell+1} - A_\ell\|_F \leq (1 + \varepsilon)\|A_{\ell+1}\|_F$ , from (2.13) we have that

$$\begin{aligned} \|A_L - A_0\|_F &= \left\| \sum_{\ell=0}^{L-1} (A_{\ell+1} - A_\ell) \right\|_F \leq \sum_{\ell=0}^{L-1} \|A_{\ell+1} - A_\ell\|_F \leq \varepsilon \sum_{\ell=0}^{L-1} \|A_{\ell+1}\|_F \\ &\leq \varepsilon \sum_{\ell=0}^{L-1} (1 + \varepsilon)^{L-\ell-1} \|A_L\|_F = [(1 + \varepsilon)^L - 1] \|A_L\|_F. \end{aligned}$$

Observing

$$\begin{aligned} \|A - A_0\|_F &\leq \|A - A_L\|_F + \|A_L - A_0\|_F \leq \varepsilon \|A\|_F + [(1 + \varepsilon)^L - 1] \|A_L\|_F \\ &\leq \varepsilon \|A\|_F + [(1 + \varepsilon)^L - 1] (\|A - A_L\|_F + \|A\|_F) \\ &\leq \{\varepsilon + (1 + \varepsilon)[(1 + \varepsilon)^L - 1]\} \|A\|_F, \end{aligned}$$

we obtain the assertion.  $\square$

The computational cost of the coarsening procedure is estimated in the following lemma.

**Lemma 2.22.** *The resulting rank of  $A_0$  is bounded by  $c_{\text{sp}} k_{\max} L(T_{I \times J})$ . Hence, the required costs are of the order*

$$c_{\text{sp}}^3 k_{\max}^2 L^3(T_{I \times J})[|I| + |J|],$$

where  $k_{\max} := \max_{t \times s \in \mathcal{L}(T_{I \times J})} k_{t \times s}$ .

*Proof.* Similarly to the case of a blockwise constant rank, the costs of coarsening  $T_{I \times J}$  can be estimated to be bounded by

$$\sum_{t \times s \in T_{I \times J}} k_{t \times s}^2 (|t| + |s|),$$

where we have omitted terms which do depend neither on  $|t|$  nor on  $|s|$ . Using (2.14), the cost of each block  $t \times s \in T_{I \times J} \setminus \mathcal{L}(T_{I \times J})$  can be estimated by a sum over its leaves

$$k_{t \times s} (|t| + |s|) \leq \sum_{t' \times s' \in \mathcal{L}(T_{t \times s})} k_{t' \times s'} (|t'| + |s'|).$$

From (1.36) it follows that  $k_{t \times s} \leq c_{\text{sp}} L(T_{t \times s}) k_{\max}$ . With the previous estimate we obtain

$$\begin{aligned} \sum_{t \times s \in T_{I \times J}} k_{t \times s}^2 (|t| + |s|) &\leq c_{\text{sp}}^2 L^2(T_{I \times J}) k_{\max}^2 \sum_{t \times s \in T_{I \times J}} (|t| + |s|) \\ &\leq c_{\text{sp}}^3 L^3(T_{I \times J}) k_{\max}^2 [|I| + |J|] \end{aligned}$$

due to (1.36).  $\square$

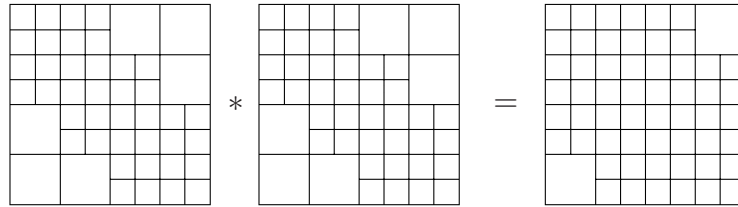
We have seen that each block  $t \times s \in \mathcal{L}(T'_{I \times J})$  of the final partition  $\mathcal{L}(T'_{I \times J})$  requires  $\mathcal{O}(k_{\max}^2(|t| + |s|))$  operations for its computation and that its accuracy is of the order  $L(T_{I \times J})\varepsilon$ . Returning to the whole matrix, the coarsening process therefore gives back a matrix  $\tilde{A}$  which has accuracy  $L(T_{I \times J})\varepsilon$  and can be computed with  $\mathcal{O}(k_{\max}^2(|I| + |J|))$  arithmetic operations; cf. (1.36).

## 2.7 Multiplying $\mathcal{H}$ -Matrices

Let  $A \in \mathcal{H}(T_{I \times J}, k_A)$  and  $B \in \mathcal{H}(T_{J \times K}, k_B)$  be two hierarchical matrices. The aim of this section is to investigate the product  $AB \in \mathbb{C}^{I \times K}$  of  $A$  and  $B$ . In contrast to the hierarchical addition, which preserves the block structure, the exact product  $AB$  cannot be represented using the block cluster tree  $T_{I \times K}$ , i.e., it cannot be guaranteed that the blockwise rank is bounded in general. Therefore, in the first part of this section we will define the product tree  $T_{IJK}$ , which is suitable to hold the exact product  $AB$ . The second part of this section is devoted to the rounded multiplication to a given partition and given rank. Our presentation mainly relies on [116].

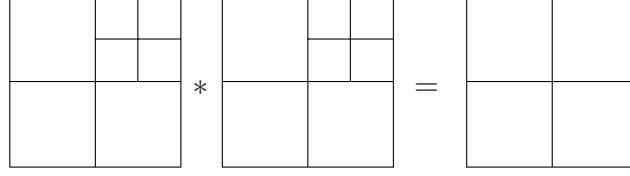
### 2.7.1 Product Block Cluster Tree

In this section we assume that  $T_{I \times J}$  has been generated using the cluster trees  $T_I$  and  $T_J$  and that for the construction of  $T_{J \times K}$  the cluster trees  $T_J$  and  $T_K$  have been used.



**Fig. 2.10** The product of two partitions.

Figure 2.10 shows that the block structure is usually not preserved. The block in the upper right corner of the product is a sum of products in which at least one factor is a low-rank matrix. Hence, its rank is bounded by the number of products times the maximum rank of blocks in  $A$  and  $B$ . The block on the left of the latter needs to be refined since for its computation a product of two factors which are not in the set of leaves is involved. The impression that the product partition is always finer than the partition of the factors is wrong as can be seen from Fig. 2.11.



**Fig. 2.11** Factors leading to a coarser product partition.

**Definition 2.23.** The **product tree**  $T_{IJK}$  of  $T_{I \times J}$  and  $T_{J \times K}$  is inductively defined by

- (i)  $I \times K$  is the root of  $T_{IJK}$
- (ii) The set of sons of blocks  $t \times s \in T_{IJK}$  from the  $\ell$ th level of  $T_{IJK}$  is

$$S_{IJK}(t \times s) := \left\{ t' \times s' \mid \exists r \in T_J^{(\ell)}, r' \in T_J^{(\ell+1)} : t' \times r' \in S_{I \times J}(t \times r) \text{ and } r' \times s' \in S_{J \times K}(r \times s) \right\}.$$

**Lemma 2.24.** The product tree  $T_{IJK}$  is a block cluster tree based on  $T_I$  and  $T_K$ . For its depth it holds that

$$L(T_{IJK}) \leq \min\{L(T_{I \times J}), L(T_{J \times K})\}.$$

The sparsity constant of  $T_{IJK}$  can be estimated as

$$c_{\text{sp}}(T_{IJK}) \leq c_{\text{sp}}(T_{I \times J}) \cdot c_{\text{sp}}(T_{J \times K}).$$

*Proof.* Due to Definition 2.23 it holds that for a given  $t \in T_I$

$$\{s \in T_K : t \times s \in T_{IJK}\} \subset \{s \in T_K \mid \exists r \in T_J : t \times r \in T_{I \times J} \text{ and } r \times s \in T_{J \times K}\}.$$

Therefore, recalling Definition 1.35 we have that

$$\begin{aligned} |\{s \in T_K : t \times s \in T_{IJK}\}| &\leq \sum_{r \in T_J : t \times r \in T_{I \times J}} |\{s \in T_K : r \times s \in T_{J \times K}\}| \\ &\leq c_{\text{sp}}(T_{I \times J}) \cdot c_{\text{sp}}(T_{J \times K}). \end{aligned}$$

The rest of the assertion is an easy consequence of Definition 2.23.  $\square$

Let  $t \times s \in T_{IJK}$  be a leaf from the  $\ell$ th level of  $T_{IJK}$ . Then

$$(AB)_{ts} = \sum_{j \in J} A_{tj} B_{js}.$$

We will rearrange the previous summation to a sum of products in which one factor is a low-rank matrix. To this end, denote by  $F_j(t) \in T_I$  and  $F_j(s) \in T_K$  the uniquely defined ancestors of  $t$  and  $s$  from the  $j$ th,  $0 \leq j \leq \ell$ , level of  $T_I$  and  $T_K$ , respectively. We define the set

$$U_j(t \times s) := \left\{ r \in T_J^{(j)} : F_j(t) \times r \in T_{I \times J} \text{ and } r \times F_j(s) \in \mathcal{L}(T_{J \times K}) \right. \\ \left. \text{or } F_j(t) \times r \in \mathcal{L}(T_{I \times J}) \text{ and } r \times F_j(s) \in T_{J \times K} \right\}.$$

The consequence of the following lemma is that

$$(AB)_{ts} = \sum_{j=0}^{\ell} \sum_{r \in U_j(t \times s)} A_{tr} B_{rs}. \quad (2.15)$$

**Lemma 2.25.** *It holds that*

$$\bigcup_{j=0}^{\ell} \bigcup_{r \in U_j(t \times s)} r = J,$$

where the union is pairwise disjoint. Furthermore, it holds that

$$|U_j(t \times s)| \leq \min\{c_{\text{sp}}(T_{I \times J}), c_{\text{sp}}(T_{J \times K})\}, \quad 0 \leq j \leq \ell.$$

*Proof.* Let  $v \in J$ . It holds that  $b_0 := F_0(t) \times J \in T_{I \times J}$  and  $b'_0 := J \times F_0(s) \in T_{J \times K}$ . If neither  $b_0$  nor  $b'_0$  is a leaf, then there is  $r_1 \in S_J(J)$  such that  $v \in r_1$  and  $b_1 := F_1(t) \times r_1 \in T_{I \times J}$ ,  $b'_1 := r_1 \times F_1(s) \in T_{J \times K}$ . If still neither  $b_1$  nor  $b'_1$  is a leaf, we descend the trees keeping  $v \in r_j$  until for some  $j$  either  $b_j := F_j(t) \times r_j$  or  $b'_j := r_j \times F_j(s)$  is a leaf. In this case  $v \in r_j \in U_j$ . Since  $t \times s$  is a leaf in  $T_{IJK}$ , it follows that  $j \leq \ell$ .

Since  $U_j$  is constructed from  $T_J^{(j)}$ , the elements of each  $U_j$  are pairwise disjoint. Let  $r \in U_j$  and  $r' \in U_{j'}$ ,  $j \leq j'$ , and  $r \cap r' \neq \emptyset$ . Since  $r, r' \subset T_J$ , we obtain  $r' \subset r$ . It follows that

$$F_{j'}(t) \times r' \subset F_j(t) \times r \quad \text{and} \quad r' \times F_{j'}(s) \subset r \times F_j(s). \quad (2.16)$$

The definition of  $U_j$  implies that either  $F_j(t) \times r$  or  $r \times F_j(s)$  is a leaf. Hence, one of the inclusions in (2.16) is an equality, which implies that  $j = j'$  and hence that  $r = r'$ .

From

$$|U_j| \leq |\{r \in T_J^{(j)} : F_j(t) \times r \in T_{I \times J}\}| \leq c_{\text{sp}}(T_{I \times J})$$

and

$$|U_j| \leq |\{r \in T_J^{(j)} : r \times F_j(s) \in T_{J \times K}\}| \leq c_{\text{sp}}(T_{J \times K})$$

we obtain the estimate on the cardinality of  $U_j$ . □

**Theorem 2.26.** *Let  $L = L(T_{IJK})$ . For the product  $AB$  of two matrices  $A \in \mathcal{H}(T_{I \times J}, k_A)$  and  $B \in \mathcal{H}(T_{J \times K}, k_B)$  it holds that  $AB \in \mathcal{H}(T_{IJK}, k)$ , where*

$$k \leq L \min\{c_{\text{sp}}(T_{I \times J}), c_{\text{sp}}(T_{J \times K})\} \max\{k_A, k_B, n_{\min}\}.$$

The matrix  $AB$  can be computed with at most

$$c_{\text{sp}}(T_{IJK}) L \max\{k'_B N_{\text{MV}}(A), k'_A N_{\text{MV}}(B)\}$$

arithmetic operations. Here,  $N_{\text{MV}}(A)$  denotes the number of arithmetic operations required for the matrix-vector multiplication (see Sect. 2.2),  $k'_A := \max\{k_A, n_{\min}\}$ , and  $k'_B := \max\{k_B, n_{\min}\}$ .

*Proof.* Let  $t \times s \in \mathcal{L}(T_{IJK})$  be from the  $\ell$ th level of  $T_{IJK}$ . Due to (2.15) we can express  $(AB)_{ts}$  by the sum over  $L \max_{j=0, \dots, \ell} |U_j(t \times s)|$  matrix products. From the definition of  $U_j(t \times s)$  and from  $t \times r \subset F_j(t) \times r$  and  $r \times s \subset r \times F_j(s)$  we see that one of the factors of each product corresponds to a leaf and so its rank is bounded by  $\max\{k_A, k_B, n_{\min}\}$ . As a consequence,

$$k \leq L \max_{j=0, \dots, \ell} |U_j(t \times s)| \max\{k_A, k_B, n_{\min}\}.$$

The first part of the assertion follows from Lemma 2.25.

Using the representation (2.15), we have to compute the products  $A_{tr}B_{rs}$ , each of which consists of  $\max\{k_A, k_B, n_{\min}\}$  matrix-vector products. Hence, with  $P := \mathcal{L}(T_{IJK})$  and  $P_i := P \cap T_{IJK}^{(i)}$ ,  $0 \leq i < L$ , we obtain for the number of arithmetic operations for the matrix product

$$\begin{aligned} N_{\text{MM}}(A, B) &\leq \sum_{t \times s \in P} \sum_{j=0}^{L-1} \sum_{r \in U_j(t \times s)} \max\{k'_B N_{\text{MV}}(A_{tr}), k'_A N_{\text{MV}}(B_{rs})\} \\ &\leq \sum_{t \times s \in P} \max\{k'_B N_{\text{MV}}(A_{tJ}), k'_A N_{\text{MV}}(B_{Js})\} \\ &\leq \sum_{i=0}^{L-1} \sum_{t \times s \in P_i} \max\{k'_B N_{\text{MV}}(A_{tJ}), k'_A N_{\text{MV}}(B_{Js})\} \\ &\leq c_{\text{sp}}(T_{IJK}) L \max\{k'_B N_{\text{MV}}(A), k'_A N_{\text{MV}}(B)\}, \end{aligned}$$

which proves the assertion.  $\square$

### 2.7.2 Preserving the Original Block Structure

The exact product  $AB$  of two  $\mathcal{H}$ -matrices  $A \in \mathcal{H}(T_{I \times J}, k_A)$  and  $B \in \mathcal{H}(T_{J \times K}, k_B)$  can be found in the set  $\mathcal{H}(T_{IJK}, k')$  with a slightly increased blockwise rank  $k'$  as we have just seen in Theorem 2.26. Since the product tree  $T_{IJK}$  will usually lead to a finer partition, which in turn leads to an increased numerical effort, the product  $AB$  is preferably represented on the usual partition of the cluster tree  $T_{I \times K}$  with possibly further increased rank. By the following *idempotency constant* it is possible to estimate this increment. For simplicity we restrict ourselves to the case  $I = J = K$ .

**Definition 2.27.** Let  $T_{I \times I}$  be a block cluster tree generated from the cluster tree  $T_I$ . The **idempotency constant**  $c_{\text{id}}$  is defined as

$$c_{\text{id}}(b) := |\{t' \times s' \in T_{I \times I} : t' \times s' \subset b \text{ and } \exists r' \in T_I \text{ with } t' \times r', r' \times s' \in T_{I \times I}\}|$$

for  $b \in \mathcal{L}(T_{I \times I})$  and

$$c_{\text{id}} := \max_{b \in \mathcal{L}(T_{I \times I})} c_{\text{id}}(b).$$

If  $T_{III}$  is not finer than  $T_{I \times I}$ , then  $c_{\text{id}} = 1$ . In Fig. 2.10 four blocks are refined into its four sons, respectively. In this case it holds that  $c_{\text{id}} = 5$ .

*Example 2.28.* In Example 1.36 we have estimated the sparsity constant  $c_{\text{sp}}$  under the assumption (see (1.22)) that

$$(\text{diam } X_t)^m \leq c_g 2^{-\ell} \quad \text{and} \quad \mu(X_t) \geq 2^{-\ell}/c_G$$

for all  $t \in T_I^{(\ell)}$ . The same assumption will now be used to estimate the idempotency constant. Let  $b \in \mathcal{L}(T_{I \times I})$  be from the  $\ell$ th level of  $T_{I \times I}$ . If  $b$  is a non-admissible leaf, then  $c_{\text{id}}(b) = 1$ . For admissible  $b = t \times s$  we define

$$q := \log_2(c_g c_G c_\Omega) + m \log_2(2 + \eta),$$

where  $c_\Omega$  is defined in (1.19). We will show that for clusters  $t', r', s' \in T_I$ ,  $t' \times s' \subset b = t \times s$ , satisfying  $t' \times r', r' \times s' \in T_{I \times I}^{(\ell+q)}$  it follows that either  $t' \times r'$  or  $r' \times s'$  is a leaf in  $T_{I \times I}$ . This can be seen from

$$2^{-q/m} = c_g^{-1/m} c_G^{-1/m} c_\Omega^{-1/m} (2 + \eta)^{-1}$$

and

$$\begin{aligned} \text{diam } X_{r'} &= (1 + \eta/2) \text{diam } X_{r'} - \eta/2 \text{diam } X_{r'} \\ &\leq (1 + \eta/2) c_g^{1/m} 2^{-(\ell+q)/m} - \eta/2 \text{diam } X_{r'} \\ &\leq c_\Omega^{-1/m} / 2 \min\{\mu^{1/m}(X_t), \mu^{1/m}(X_s)\} - \eta/2 \text{diam } X_{r'} \\ &\leq \frac{1}{2} \min\{\text{diam } X_t, \text{diam } X_s\} - \eta/2 \text{diam } X_{r'} \\ &\leq \frac{\eta}{2} (\text{dist}(X_t, X_s) - \text{diam } X_{r'}) \\ &\leq \eta \max\{\text{dist}(X_{t'}, X_{r'}), \text{dist}(X_{r'}, X_{s'})\}. \end{aligned}$$

Since hence either  $t' \times r'$  or  $r' \times s'$  is admissible, one of these blocks does not have descendants in  $T_{I \times I}$ . Hence, the number of vertices in  $T_{III}$  which are contained in  $b$  is bounded by  $c_{\text{id}} \leq 4^q = (c_g c_G c_\Omega)^2 (2 + \eta)^{2m}$ .

The same kind of proof can be adapted to partitions generated from algebraic clustering.

**Lemma 2.29.** *Under the assumption (1.28) it holds that  $c_{\text{id}} \leq [c_u(2 + \eta)]^{2m}$ .*

*Proof.* Let  $b \in \mathcal{L}(T_{I \times I})$  be from the  $\ell$ th level of  $T_{I \times I}$ . If  $b$  is a non-admissible leaf, then  $c_{\text{id}}(b) = 1$ . For admissible  $b = t \times s$  we define

$$q := m \log_2 c_u (2 + \eta).$$

We will show that for clusters  $t', r', s' \in T_I$ ,  $t' \times s' \subset b = t \times s$ , satisfying  $t' \times r', r' \times s' \in T_{I \times I}^{(\ell+q)}$  it follows that either  $t' \times r'$  or  $r' \times s'$  is a leaf in  $T_{I \times I}$ . This can be seen from

$$\begin{aligned} \text{diam } r' &= (1 + \eta/2) \text{diam } r' - \eta/2 \text{diam } r' \\ &\leq (1 + \eta/2) c_u 2^{-q/m} \min\{\text{diam } t, \text{diam } s\} - \eta/2 \text{diam } r' \\ &\leq \frac{1}{2} \min\{\text{diam } t, \text{diam } s\} - \eta/2 \text{diam } r' \\ &\leq \frac{\eta}{2} (\text{dist}(t, s) - \text{diam } r') \\ &\leq \eta \max\{\text{dist}(t', r'), \text{dist}(r', s')\}. \end{aligned}$$

Since hence either  $t' \times r'$  or  $r' \times s'$  is admissible, one of these blocks does not have descendants in  $T_{I \times I}$ . Therefore, the number of vertices which are contained in  $b$  is bounded by  $c_{\text{id}} \leq 4^q = [c_u(2 + \eta)]^{2m}$ .  $\square$

**Theorem 2.30.** *Let  $A, B \in \mathcal{H}(T_{I \times I}, k)$ . Then  $AB \in \mathcal{H}(T_{I \times I}, \hat{k})$ , where*

$$\hat{k} \leq c_{\text{id}} c_{\text{sp}} L(T_{I \times I}) \max\{k, n_{\min}\}.$$

*Proof.* Due to Theorem 2.26, we have  $AB \in \mathcal{H}(T_{III}, k')$ , where the blockwise rank  $k'$  is bounded by  $c_{\text{sp}} L(T_{I \times I}) \max\{k, n_{\min}\}$ . If a leaf from  $T_{I \times I}$  is contained in a leaf from  $T_{III}$ , then the restriction does not increase the rank. If a leaf from  $T_{I \times I}$  contains leaves from  $T_{III}$ , then their number is bounded by  $c_{\text{id}}$ . Therefore, the rank is bounded by  $c_{\text{id}} k'$ .  $\square$

### 2.7.3 Rounded Multiplication

If the product  $AB \in \mathcal{H}(T_{I \times I}, k)$  is to be approximated by a matrix from  $\mathcal{H}(T_{I \times I}, \tilde{k})$ ,  $\tilde{k} < k$ , then one of the rounding algorithms from Sect. 1.1.5 can be used to reduce the blockwise rank to  $\tilde{k}$ . The first sums up all arising products in (2.15) and rounds the result to rank  $\tilde{k}$ . As shown in Sect. 1.1.5, the result of the rounding can be controlled relatively to the best approximation. The second algorithm gradually adds the products to a rounded sum and is significantly faster but can result in arbitrarily large errors for general matrices. The following divide-and-conquer algorithm for the computation of the approximate update  $C := C + AB$  of  $C \in \mathcal{H}(T_{I \times I}, \tilde{k})$  stems from the blockwise matrix multiplication and is even faster.

Assume that  $A \in \mathcal{H}(T_{I \times J}, k_A)$  and  $B \in \mathcal{H}(T_{J \times K}, k_B)$  are subdivided according to their block cluster trees  $T_{I \times J}$  and  $T_{J \times K}$ :

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

Then  $AB$  has the following block structure

$$AB = \begin{bmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{bmatrix}.$$

If the target matrix  $C$  has sons in  $T_{I \times K}$ , then compute

$$C_{ij} := C_{ij} + A_{i1}B_{1j} + A_{i2}B_{2j}, \quad i, j = 1, 2,$$

each of which has approximately half the size of  $C := C + AB$ . In the case that  $C$  is a leaf in  $T_{I \times K}$ , the sums  $A_{i1}B_{1j} + A_{i2}B_{2j}$  are rounded to rank- $\tilde{k}$  matrices  $R_{ij}$ ,  $i, j = 1, 2$ , and

$$\begin{bmatrix} R_{11} & R_{12} \\ R_{21} & R_{22} \end{bmatrix}$$

is agglomerated to a single rank- $\tilde{k}$  matrix (see Sect. 1.1.6) before adding it to  $C$  using one of the rounded additions. The complexity of the rounded multiplication was shown to be of the order  $k^2 L^2(T_I)|I| + k^3|I|$  for  $I = J$ ; see [132]. A parallel version of the previous algorithm was presented in [166].

#### 2.7.4 Multiplication of Hierarchical and Semi-Separable Matrices

As we have seen in Sect. 1.2, multiplying a  $(p, q)$ -semi-separable matrix  $S$  and a general matrix  $A \in \mathbb{C}^{m \times n}$  can be done with  $\mathcal{O}((p+q)mn)$  arithmetic operations. If  $A$  is an  $\mathcal{H}$ -matrix, then  $AS$  and  $SA$  can be computed with significantly less effort. Since diagonal-plus-semi-separable matrices are hierarchical matrices, we could use the hierarchical matrix multiplication algorithm from the previous section. We will however present an algorithm which is significantly more efficient if one of the matrices is semi-separable. Its complexity will actually be of the order of the hierarchical matrix-vector multiplication. The resulting algorithms will be used in Sect. 4.6 when updating the factors of the  $LU$  decomposition in Broyden's method.

In the following lemma (see [24]), which is the basis for the efficient multiplication, we make use of the notation

$$s' := \{i \in I : i < \min s\} \quad \text{and} \quad s'' := \{i \in I : i > \max s\}$$

for  $s \subset I$ . Note that this lemma holds for arbitrary partitions  $P$  of the matrix indices  $\{1, \dots, m\} \times \{1, \dots, n\}$ .

**Lemma 2.31.** *Let  $A \in \mathbb{C}^{m \times n}$  and let  $S \in \mathbb{C}^{n \times n}$  be a diagonal-plus-semi-separable matrix with the notation from Definition 1.10. For  $t \times s \in P$  it holds that*

$$(AS)_{ts} = A_{ts}S_{ss} + \Xi V_s^H + Y Z_s^H,$$

where  $\Xi := A_{ts'}U_{s'} \in \mathbb{C}^{t \times p}$  and  $Y := A_{ts''}W_{s''} \in \mathbb{C}^{t \times q}$ . Furthermore, if  $A \in \mathbb{C}^{n \times m}$ , then



$$(SA)_{ts} = S_{tt}A_{ts} + U_t\hat{\Xi}^H + W_t\hat{\Upsilon}^H,$$

where  $\hat{\Xi} := A_{t''s}^H V_{t''} \in \mathbb{C}^{s \times p}$  and  $\hat{\Upsilon} := A_{t's}^H Z_{t'} \in \mathbb{C}^{s \times q}$ . Here,  $U_t$  denotes the restriction of  $U$  to the rows  $t$ .

*Proof.* By  $\ell$  we denote the level of  $t \times s$  in the block cluster tree  $T_{I \times I}$ . Let  $\tau \times \sigma \supset t \times s$  be in the  $k$ th level of  $T_{I \times I}$ . By induction over  $k \leq \ell$  we will prove that

$$A_{t\sigma}S_{\sigma s} = A_{ts}S_{ss} + \Xi(\sigma)V_s^H + \Upsilon(\sigma)Z_s^H \quad (2.17)$$

with vectors  $\Xi(\sigma) := A_{t\sigma^*}U_{\sigma^*}$  and  $\Upsilon(\sigma) = A_{t\sigma^{**}}W_{\sigma^{**}}$ , where  $\sigma^* := \{i \in \sigma : i < \min s\}$ ,  $\sigma^{**} := \{i \in \sigma : i > \max s\}$ . The choice  $\sigma = I$  will then lead to the assertion.

For  $k = \ell$  we have that  $\tau \times \sigma = t \times s$ . Therefore, we obtain

$$A_{t\sigma}S_{\sigma s} = A_{ts}S_{ss}.$$

Assume that (2.17) is true for  $k+1 \leq \ell$ . Let  $A_{\tau\sigma}$  and  $S_{\sigma\sigma}$  be partitioned corresponding to the tree  $T_{I \times I}$

$$A_{\tau\sigma} = \begin{bmatrix} A_{\tau_1\sigma_1} & A_{\tau_1\sigma_2} \\ A_{\tau_2\sigma_1} & A_{\tau_2\sigma_2} \end{bmatrix} \quad \text{and} \quad S_{\sigma\sigma} = \begin{bmatrix} S_{\sigma_1\sigma_1} & U_{\sigma_1}V_{\sigma_2}^H \\ W_{\sigma_2}Z_{\sigma_1}^H & S_{\sigma_2\sigma_2} \end{bmatrix}.$$

Then we have that

$$A_{t\sigma}S_{\sigma s} = \begin{bmatrix} A_{\tau_1\sigma_1}S_{\sigma_1\sigma_1} + A_{\tau_1\sigma_2}W_{\sigma_2}Z_{\sigma_1}^H & A_{\tau_1\sigma_2}S_{\sigma_2\sigma_2} + A_{\tau_1\sigma_1}U_{\sigma_1}V_{\sigma_2}^H \\ A_{\tau_2\sigma_1}S_{\sigma_1\sigma_1} + A_{\tau_2\sigma_2}W_{\sigma_2}Z_{\sigma_1}^H & A_{\tau_2\sigma_2}S_{\sigma_2\sigma_2} + A_{\tau_2\sigma_1}U_{\sigma_1}V_{\sigma_2}^H \end{bmatrix}_{ts}.$$

If  $s \subset \sigma_1$ , then

$$A_{t\sigma}S_{\sigma s} = A_{t\sigma_1}S_{\sigma_1 s} + A_{t\sigma_2}W_{\sigma_2}Z_s^H = A_{ts}S_{ss} + \Xi(\sigma_1)V_s^H + [\Upsilon(\sigma_1) + A_{t\sigma_2}W_{\sigma_2}]Z_s^H.$$

If, on the other hand,  $s \subset \sigma_2$ , then

$$A_{t\sigma}S_{\sigma s} = A_{t\sigma_2}S_{\sigma_2 s} + A_{t\sigma_1}U_{\sigma_1}V_s^H = A_{ts}S_{ss} + [\Xi(\sigma_2) + A_{t\sigma_1}U_{\sigma_1}]V_s^H + \Upsilon(\sigma_2)Z_s^H$$

due to the induction assumption. The second part of the assertion is obtained by similar arguments.  $\square$

According to the previous lemma, each sub-block  $(AS)_{ts}$  of  $AS$  is a rank- $(p+q)$  update of  $A_{ts}S_{ss}$ . Let  $t_1 \times s_1, \dots, t_\mu \times s_\mu$  be the blocks in  $P$  such that  $\min s_i \leq \min s_j$  for  $i \leq j$ . Then  $C := AS$  can be computed by Algorithm 2.5.

---

```

 $\Xi := 0$  from  $\mathbb{C}^{I \times p}$ 
for  $i = 1, \dots, \mu$  do
     $C_{t_i s_i} := A_{t_i s_i} S_{s_i s_i} + \Xi_{t_i} V_{s_i}^H$ 
     $\Xi_{t_i} := \Xi_{t_i} + A_{t_i s_i} U_{s_i}$ 
 $\Upsilon := 0$  from  $\mathbb{C}^{I \times q}$ 
for  $i = \mu, \dots, 1$  do
     $C_{t_i s_i} := C_{t_i s_i} + \Upsilon_{t_i} Z_{s_i}^H$ 
     $\Upsilon_{t_i} := \Upsilon_{t_i} + A_{t_i s_i} W_{s_i}$ 

```

---

Algorithm 2.5:  $\mathcal{H}$ -matrix times semi-separable matrix.

For the computation of  $SA$  the blocks have to be ordered with respect to their row indices.

In order to be able to compute  $AS$  efficiently, we have to exploit that  $A$  is an  $\mathcal{H}$ -matrix, i.e., that for each block  $t \times s \in P$  it holds that  $A_{ts} = XY^H$  with  $X \in \mathbb{C}^{t \times k}$  and  $Y \in \mathbb{C}^{s \times k}$  each consisting of  $k$  columns. In this case we have

$$A_{ts}U_s = X(Y^H U_s)$$

and each product  $A_{ts}U_s$  appearing in Algorithm 2.5 can be done with  $pk(|t| + |s|)$  operations. Additionally, products  $A_{ts}S_{ss}$  have to be computed for each block  $t \times s \in P$ . Exploiting

$$A_{ts}S_{ss} = X(S_{ss}^H Y)^H,$$

it is sufficient to compute the  $k \times s$  matrix  $S_{ss}^H Y$ , which can be done with  $\mathcal{O}((p+q)k|s|)$  operations using Algorithm 1.1. The rank- $p$  update of  $A_{ts}S_{ss}$  with  $\Xi_t V_s^H$  can be done explicitly by storing the rank- $(k+p)$  matrix

$$A_{ts}S_{ss} + \Xi_t V_s^H = [X, \Xi_t][S_{ss}^H Y, V_s]^H,$$

which requires copying  $p(|t| + |s|)$  units of storage. The updates with  $\Xi_t V_s^H$  and  $\Upsilon_t Z_s^H$  lead to a blockwise rank of  $k + p + q$ , i.e., with  $A \in \mathcal{H}(P, k)$  it holds that  $AS, SA \in \mathcal{H}(P, k + p + q)$ .

We will apply this multiplication to problems (see Sect. 4.6.2) where  $p$  and  $q$  are constants. Hence, the computational complexity of each block  $t \times s$  is of the order  $k(|t| + |s|)$ , which is exactly the complexity that is required for each block when multiplying an  $\mathcal{H}(P, k)$ -matrix by a vector. The latter multiplication requires  $\mathcal{O}(kn \log n)$  operations; cf. Sect. 2.2.

*Remark 2.32.* (a) For the update of the  $LU$  decomposition we will have to compute the product of triangular hierarchical and triangular semi-separable matrices. In this case the product will be in  $\mathcal{H}(P, k + p)$  and  $\mathcal{H}(P, k + q)$ , respectively, if the hierarchical factor is in  $\mathcal{H}(P, k)$ . It is obvious how to simplify and accelerate Algorithm 2.5 for such kind of matrices; see Algorithms 2.6 and 2.7.

(b) The rank- $p$  and rank- $q$  updates will gradually increase the rank of the factors if they are stored explicitly. This can be avoided by truncation to rank- $k$ ; see Sect. 1.1.4. The latter operation requires  $\mathcal{O}(k^2(|t| + |s|))$  operations for each block  $t \times s$ .

---

Let  $\mathcal{L} = \{t_1 \times s_1, \dots, t_\mu \times s_\mu\}$  be the blocks in  $P$  such that  $\min s_i \geq \min s_j$  for  $i \leq j$ .  
 $\Upsilon := 0$  from  $\mathbb{C}^{I \times q}$   
**for**  $i = 1, \dots, \mu$  **do**  
     $C_{t_i s_i} := L_{t_i s_i} L_{s_i s_i}^H + \Upsilon_{t_i} Z_{s_i}^H$   
     $\Upsilon_{t_i} := \Upsilon_{t_i} + L_{t_i s_i} W_{s_i}$

---

Algorithm 2.6:  $\mathcal{H}$ -matrix times lower triangular semi-separable matrix.

---

Let  $\mathcal{L} = \{t_1 \times s_1, \dots, t_\mu \times s_\mu\}$  be the blocks in  $P$  such that  $\min t_i \geq \min t_j$  for  $i \leq j$ .  
 $\Xi := 0$  from  $\mathbb{C}^{I \times P}$   
**for**  $i = 1, \dots, \mu$  **do**  
     $C_{t_i s_i} := U'_{t_i} U_{t_i s_i} + U_{t_i} \Xi_{s_i}^H$   
     $\Xi_{s_i} := \Xi_{s_i} + U_{t_i s_i}^H V_{t_i}$

---

Algorithm 2.7: Upper triangular semi-separable matrix times  $\mathcal{H}$ -matrix.

## 2.8 Hierarchical Inversion

In this section we assume that each block  $A_t$ ,  $t \in T_I$ , of  $A \in \mathcal{H}(T_{I \times I}, k)$  is invertible. Positive definite matrices are an important example.

Two approaches to the computation of the  $\mathcal{H}$ -matrix inverse have been investigated in the literature. The first (see [132]) uses the **Schulz iteration**

$$C_{i+1} = C_i(2I - AC_i), \quad i = 0, 1, 2, \dots,$$

which arises from Newton's method applied to  $F(X) := X^{-1} - A = 0$ . This iteration converges locally to  $A^{-1}$ ; see [235]. The following divide-and-conquer approach (see [127]) has turned out to be significantly more efficient. It exploits the property that each matrix  $A \in \mathcal{H}(T_{I \times I}, k)$  is subdivided according to its block cluster tree:

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

It is easy to see that for the exact inverse of  $A$  it holds that

$$A^{-1} = \begin{bmatrix} A_{11}^{-1} + A_{11}^{-1} A_{12} S^{-1} A_{21} A_{11}^{-1} & -A_{11}^{-1} A_{12} S^{-1} \\ -S^{-1} A_{21} A_{11}^{-1} & S^{-1} \end{bmatrix}, \quad (2.18)$$

where  $S$  denotes the Schur complement  $S := A_{22} - A_{21} A_{11}^{-1} A_{12}$  of  $A_{11}$  in  $A$ . For the computation of  $A^{-1}$  the matrices  $A_{11}$  and  $S$ , which have approximately half the size of  $A$ , have to be inverted. The  $\mathcal{H}$ -matrix inverse  $C$  of  $A$  is computed by replacing the multiplications and the additions appearing in (2.18) by the  $\mathcal{H}$ -matrix versions. We need a temporary matrix  $T \in \mathcal{H}(T_{I \times I}, k)$ , which together with  $C$  is initialized to zero.

---

```

procedure invertH( $t, A, \text{var } C$ )
if  $t \in \mathcal{L}(T_I)$  then  $C_{tt} := A_{tt}^{-1}$  is the usual inverse.
else
  let  $t_1, t_2$  denote the sons of  $t$ .
  invertH( $t_1, A, C$ ).
   $T_{t_1 t_2} = T_{t_1 t_2} - C_{t_1 t_1} A_{t_1 t_2}$ .
   $T_{t_2 t_1} = T_{t_2 t_1} - A_{t_2 t_1} C_{t_1 t_1}$ .
   $A_{t_2 t_2} = A_{t_2 t_2} + A_{t_2 t_1} T_{t_1 t_2}$ .
  invertH( $t_2, A, C$ ).
   $C_{t_1 t_2} = C_{t_1 t_2} + T_{t_1 t_2} C_{t_2 t_2}$ .
   $C_{t_2 t_1} = C_{t_2 t_1} + C_{t_2 t_2} T_{t_2 t_1}$ .
   $C_{t_1 t_1} = C_{t_1 t_1} + T_{t_1 t_2} C_{t_2 t_1}$ .

```

---

Algorithm 2.8:  $\mathcal{H}$ -matrix inversion.

Calling `invertH` with parameters  $I$ ,  $A$ , and  $C$  generates the desired approximation  $C$  of  $A^{-1}$ , while  $A$  is destroyed.

The complexity of the computation of the  $\mathcal{H}$ -inverse is determined by the cost of the  $\mathcal{H}$ -matrix multiplication. For the following theorem it is assumed that each sum of two rank- $k$  matrices arising during the previous algorithm is rounded to rank  $k$ . We remark that the feasibility of this assumption has to be checked for the respective problem since otherwise the approximation error may become uncontrollable.

**Theorem 2.33.** *The computation of the  $\mathcal{H}$ -matrix inverse  $C \in \mathcal{H}(T_{I \times I}, k)$  of  $A \in \mathcal{H}(T_{I \times I}, k)$  using Algorithm 2.8 requires  $\mathcal{O}(k^2 L^2(T_I) |I|)$  operations.*

If we prescribe the rounding precision  $\varepsilon_{\mathcal{H}}$ , it is by no means obvious that the required blockwise rank  $k$  of the  $\mathcal{H}$ -matrix inverse  $C$  is small. In order to prove this, we will leave our algebraic point of view and exploit analytic properties which are accessible for subclasses of matrices such as those arising from elliptic operators; see Chap. 4.

The  $\mathcal{H}$ -matrix inverse may be used for the data-sparse approximation of operator valued functions. Let  $f : \mathbb{C} \rightarrow \mathbb{C}$  be analytic inside of a path  $\Gamma \subset \mathbb{C}$  enveloping the spectrum of an elliptic operator  $\mathcal{L}$ . The operator  $f(\mathcal{L})$  can be represented using the **Dunford-Cauchy integral**

$$f(\mathcal{L}) = \frac{1}{2\pi i} \int_{\Gamma} f(z)(zI - \mathcal{L})^{-1} dz$$

provided that this integral converges. Approximating the previous integral by appropriate quadrature formula and treating the discrete resolvents by the  $\mathcal{H}$ -matrix inverse leads to a data-sparse approximation. Examples are the **operator exponential**  $\exp(-t\mathcal{L})$  arising from the solution of the heat equation and the **operator cosine family**  $\cos(t\sqrt{\mathcal{L}})$  arising from the wave equation; see [98, 96]. In addition, data-sparse methods for the approximation of the Sylvester and the Riccati equation including the matrix sign-function are investigated in [96, 97].

### 2.8.0.1 Updates of the Inverse

Sometimes, many systems with coefficient matrices differing in only a small number of entries have to be solved. This problem arises, for instance, as a consequence of Newton's method for the solution of nonlinear systems if the coefficients of the operator change only locally. Assume that  $A$  is invertible such that  $1 + \alpha e_j^T A^{-1} e_i \neq 0$  with  $\alpha \in \mathbb{R}$  and the canonical vectors  $e_i, e_j \in \mathbb{R}^n$ . Due to the Sherman-Morrison-Woodbury formula (1.3) for the inverse of

$$\tilde{A} := A + \alpha e_i e_j^T$$

it holds that

$$\tilde{A}^{-1} = (A + \alpha e_i e_j^T)^{-1} = A^{-1} - \frac{\alpha}{1 + \alpha e_j^T A^{-1} e_i} A^{-1} e_i e_j^T A^{-1}.$$

Hence,  $\tilde{A}^{-1}$  and  $A^{-1}$  differ only by matrix of rank 1. In the case that  $A$  and  $\tilde{A}$  have  $r$  different entries, the update will be of rank at most  $r$ . Using the  $\mathcal{H}$ -matrix addition, we are able to compute an approximation  $C \in \mathcal{H}(T_{I \times I}, k)$  of  $\tilde{A}^{-1}$  exploiting the previously computed approximation of  $A^{-1}$ . Obviously, this is much faster than computing the  $\mathcal{H}$ -matrix inverse of  $\tilde{A}$  from scratch.

## 2.9 Computing the $\mathcal{H}$ -Matrix $LU$ Decomposition

Although the hierarchical inversion has almost linear complexity provided that the required blockwise rank behaves well, the numerical effort for its computation is still relatively high. The following hierarchical  $LU$  decomposition provides a significantly more efficient alternative.

The first algorithm for the computation of hierarchical  $LU$  decompositions has been proposed in [176]. This algorithm, however, was defined on a partition which is too restrictive to treat problems of higher spatial dimension. The following algorithm is the first method (see [21]) which can be applied to general  $\mathcal{H}$ -matrices. Once again, the required blockwise rank cannot be predicted without restricting the class of matrices. For elliptic operators it will be possible to prove a logarithmic dependence on  $|I|$ ; cf. Sect. 4.3.

We have seen that approximate versions of the usual matrix operations like addition and multiplication can be defined on the set  $\mathcal{H}(T_{I \times I}, k)$  of hierarchical matrices. The hierarchical  $LU$  decomposition can be computed using these accelerated operations during the block  $LU$  decomposition instead of the usual ones. The rounding precision these operations are performed with will be denoted by  $\varepsilon_{\mathcal{H}}$ .

To define the  $\mathcal{H}$ -matrix  $LU$  decomposition, we exploit the hierarchical block structure of a block  $A_{tt}$ ,  $t \in T_I \setminus \mathcal{L}(T_I)$ :

$$A_{tt} = \begin{bmatrix} A_{t_1 t_1} & A_{t_1 t_2} \\ A_{t_2 t_1} & A_{t_2 t_2} \end{bmatrix} = \begin{bmatrix} L_{t_1 t_1} & \\ & L_{t_2 t_1} \ L_{t_2 t_2} \end{bmatrix} \begin{bmatrix} U_{t_1 t_1} & U_{t_1 t_2} \\ & U_{t_2 t_2} \end{bmatrix},$$

where  $t_1, t_2 \in T_I$  denote the sons of  $t$  in  $T_I$ . Hence, the  $LU$  decomposition of a block  $A_{tt}$  is reduced to the following four problems on the sons of  $t \times t$ :

- (i) Compute  $L_{t_1 t_1}$  and  $U_{t_1 t_1}$  from the  $LU$  decomposition  $L_{t_1 t_1} U_{t_1 t_1} = A_{t_1 t_1}$ ;
- (ii) Compute  $U_{t_1 t_2}$  from  $L_{t_1 t_1} U_{t_1 t_2} = A_{t_1 t_2}$ ;
- (iii) Compute  $L_{t_2 t_1}$  from  $L_{t_2 t_1} U_{t_1 t_1} = A_{t_2 t_1}$ ;
- (iv) Compute  $L_{t_2 t_2}$  and  $U_{t_2 t_2}$  from the  $LU$  decomposition  $L_{t_2 t_2} U_{t_2 t_2} = A_{t_2 t_2} - L_{t_2 t_1} U_{t_1 t_2}$ .

If a block  $t \times t \in \mathcal{L}(T_{I \times I})$  is a leaf, the usual pivoted  $LU$  decomposition is employed. For (i) and (iv) two  $LU$  decompositions of half the size have to be computed. In order to solve (ii), i.e., solve a problem of the structure  $L_{tt} B_{ts} = A_{ts}$  for  $B_{ts}$ , where  $L_{tt}$  is a lower triangular matrix and  $t \times s \in T_{I \times I}$ , we use the following recursive block forward substitution. If the block  $t \times s$  is not a leaf in  $T_{I \times I}$ , from the subdivision of the blocks  $A_{ts}$ ,  $B_{ts}$ , and  $L_{tt}$  into their sub-blocks ( $t_1, t_2$  and  $s_1, s_2$  are again the sons of  $t$  and  $s$ , respectively)

$$\begin{bmatrix} L_{t_1 t_1} & \\ & L_{t_2 t_1} \ L_{t_2 t_2} \end{bmatrix} \begin{bmatrix} B_{t_1 s_1} & B_{t_1 s_2} \\ B_{t_2 s_1} & B_{t_2 s_2} \end{bmatrix} = \begin{bmatrix} A_{t_1 s_1} & A_{t_1 s_2} \\ A_{t_2 s_1} & A_{t_2 s_2} \end{bmatrix}$$

one observes that  $B_{ts}$  can be found from the following equations

$$\begin{aligned} L_{t_1 t_1} B_{t_1 s_1} &= A_{t_1 s_1}, \\ L_{t_1 t_1} B_{t_1 s_2} &= A_{t_1 s_2}, \\ L_{t_2 t_2} B_{t_2 s_1} &= A_{t_2 s_1} - L_{t_2 t_1} B_{t_1 s_1}, \\ L_{t_2 t_2} B_{t_2 s_2} &= A_{t_2 s_2} - L_{t_2 t_1} B_{t_1 s_2}, \end{aligned}$$

which are again of type (ii). If, on the other hand,  $t \times s$  is a leaf, then the usual forward substitution is applied. Similarly, one can solve (iii) by recursive block backward substitution.

The complexity of the above recursions is determined by the complexity of the hierarchical matrix multiplication, which can be estimated as  $\mathcal{O}(k^2 L^2(T_I)|I|)$  for two matrices from  $\mathcal{H}(T_{I \times I}, k)$ . Each operation is carried out with precision  $\varepsilon_{\mathcal{H}}$ . A result [79] on the stability analysis of the block  $LU$  decomposition states that the product  $LU$  is backward stable in the sense that

$$\|A - LU\|_2 < c(|I|)\varepsilon_{\mathcal{H}}(\|A\|_2 + \|L\|_2\|U\|_2).$$

Provided that  $\|L\|_2\|U\|_2 \approx \|A\|_2$ , the relative accuracy of  $LU$  will hence be of order  $\varepsilon_{\mathcal{H}}$ . Employing the  $\mathcal{H}$ -matrix arithmetic, it is therefore possible to generate an approximate  $LU$  decomposition of a matrix  $A \in \mathcal{H}(T_{I \times I}, k)$  to any prescribed accuracy with almost linear complexity whenever the blockwise rank is guaranteed to be logarithmically bounded. A logarithmic dependence of  $k$  on  $|I|$  will, for instance, be proved under quite general assumptions for finite element stiffness matrices arising from the discretization of elliptic boundary value problems.

It is known that the pointwise  $LU$  decomposition preserves the bandwidth and the skyline structure of a matrix. This property is obviously inherited by the  $\mathcal{H}$ -matrix  $LU$  decomposition. In Sect. 4.5 we will exploit this in the context of nested dissection reorderings, which in particular allows to parallelize the  $LU$  factorization algorithm.

In the case of positive definite matrices  $A$  it is possible to define an  $\mathcal{H}$ -matrix version of the Cholesky decomposition of a block  $A_{tt}$ ,  $t \in T_I \setminus \mathcal{L}(T_I)$ :

$$A_{tt} = \begin{bmatrix} A_{t_1 t_1} & A_{t_1 t_2} \\ A_{t_1 t_2}^H & A_{t_2 t_2} \end{bmatrix} = \begin{bmatrix} L_{t_1 t_1} & \\ & L_{t_2 t_2} \end{bmatrix} \begin{bmatrix} L_{t_1 t_1} & \\ & L_{t_2 t_2} \end{bmatrix}^H.$$

This factorization is recursively computed by

$$\begin{aligned} L_{t_1 t_1} L_{t_1 t_1}^H &= A_{t_1 t_1}, \\ L_{t_1 t_1} L_{t_2 t_1}^H &= A_{t_1 t_2}, \\ L_{t_2 t_2} L_{t_2 t_2}^H &= A_{t_2 t_2} - L_{t_2 t_1} L_{t_2 t_1}^H \end{aligned}$$

using the usual Cholesky decomposition on the leaves of  $T_{I \times I}$ . The second equation  $L_{t_1 t_1} L_{t_2 t_1}^H = A_{t_1 t_2}$  is solved for  $L_{t_2 t_1}$  in a way similar to how  $U_{t_1 t_2}$  was previously obtained in the  $LU$  decomposition.

Once  $A \approx L_{\mathcal{H}} U_{\mathcal{H}}$  has been decomposed, the solution of  $Ax = b$  can be found by forward/backward substitution:  $L_{\mathcal{H}} y = b$  and  $U_{\mathcal{H}} x = y$ . An advantage of the inverse of a matrix  $A$  is that  $A^{-1}$  only has to be multiplied by  $b$  when solving the linear system  $Ax = b$  for  $x$ . The  $LU$  decomposition requires forward-/backward substitution, which in standard arithmetic has quadratic complexity. Since  $L_{\mathcal{H}}$  and  $U_{\mathcal{H}}$  are  $\mathcal{H}$ -matrices,  $y_t$ ,  $t \in T_I \setminus \mathcal{L}(T_I)$ , can be computed recursively by solving the following systems for  $y_{t_1}$  and  $y_{t_2}$

$$L_{t_1 t_1} y_{t_1} = b_{t_1} \quad \text{and} \quad L_{t_2 t_2} y_{t_2} = b_{t_2} - L_{t_2 t_1} y_{t_1}.$$

If  $t \in \mathcal{L}(T_I)$  is a leaf, a usual triangular solver is used. The backward substitution can be done analogously. These substitutions are exact and their complexity is determined by the complexity of the hierarchical matrix-vector multiplication, which is  $\mathcal{O}(kL(T_I)|I|)$  for multiplying an  $\mathcal{H}(T_{I \times I}, k)$ -matrix by a vector.

## 2.10 Hierarchical $QR$ Decomposition

In addition to the hierarchical  $LU$  decomposition it seems straight forward to define a hierarchical  $QR$  decomposition of  $A \in \mathcal{H}(T_{I \times I}, k)$ , which may, for instance, be used to solve eigenvalue problems with almost linear complexity. In [176] the  $QR$  decomposition of  $\mathcal{H}$ -matrices is computed using the Cholesky decomposition  $LL^H = B$  of  $B := A^H A$ . The matrix  $Q$  is found by forward substitution from  $A = QL^H$ . Then  $Q$  is unitary because

$$Q^H Q = L^{-1} A^H A L^{-H} = L^{-1} B L^{-H} = I.$$

Since squaring a matrix should be avoided, we propose to use the following alternative recursion.

Assume that  $A$  or a diagonal sub-block of  $A$  is partitioned in the following way:

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

Setting  $X := A_{21}A_{11}^{-1}$ , it is obvious that  $I + X^H X$  and  $I + X X^H$  are Hermitian positive definite and share the same spectra (without 1). Let the Cholesky factors  $L_1$  and  $L_2$  be computed from

$$I + X^H X = L_1 L_1^H \quad \text{and} \quad I + X X^H = L_2 L_2^H,$$

then it follows that  $\det L_1 = \det L_2$ . The matrix

$$Q := \begin{bmatrix} L_1^{-1} & L_1^{-1} X^H \\ -L_2^{-1} X & L_2^{-1} \end{bmatrix} = \begin{bmatrix} L_1^{-1} & \\ & L_2^{-1} \end{bmatrix} \begin{bmatrix} I & X^H \\ -X & I \end{bmatrix}$$

is unitary and satisfies

$$\begin{aligned} QA &= \begin{bmatrix} L_1^{-1}(A_{11} + X^H A_{21}) & L_1^{-1}(A_{12} + X^H A_{22}) \\ 0 & L_2^{-1}(A_{22} - X A_{12}) \end{bmatrix} \\ &= \begin{bmatrix} L_1^H A_{11} & L_1^{-1}(A_{12} + X^H A_{22}) \\ 0 & L_2^{-1} S \end{bmatrix}, \end{aligned} \quad (2.19)$$

where  $S := A_{22} - X A_{12}$  is the Schur complement of  $A_{11}$  in  $A$ . If one of the diagonal blocks of  $QA$  is a leaf in the block cluster tree, then the usual  $QR$  decomposition is applied to it. Otherwise, two appropriate transformations of half the size have to be computed and applied. Instead of immediately applying the next transformation to the upper right block of (2.19), it is more efficient to gradually apply the transformations to each other before they are multiplied by this block. In this case, the whole  $QR$  recursion is determined by the complexity of the  $\mathcal{H}$ -matrix multiplication, which is of order  $k^2 L^2(T_I)|I|$ .

The matrix  $Q$  can be regarded as a block Givens rotation since  $\det Q = 1$ , which follows from

$$\det Q = \det \begin{bmatrix} L_1^{-1} & L_1^{-1} X^H \\ 0 & L_2^{-1}(I + X X^H) \end{bmatrix} = \det \begin{bmatrix} L_1^{-1} & L_1^{-1} X^H \\ 0 & L_2^H \end{bmatrix}.$$

Since the computation of the factors  $L_1$  and  $L_2$  involves approximation errors,  $Q$  will be only “approximately” unitary; i.e., there is an error matrix  $E \in \mathbb{C}^{I \times I}$  such that

$$(Q + E)^H (Q + E) = I.$$



Since the loss of orthogonality of the columns of  $Q$  is amplified by a bad condition number of  $A$ ,  $Q$  can be improved by decomposing  $Q = Q'R'$ , which leads to  $A = Q'\hat{R}$  with the upper block triangular matrix  $\hat{R} := R'R$ .

In order to transform  $A$  to an upper triangular matrix, we have to apply a sequence of approximately unitary matrices  $Q_1, \dots, Q_L$ , where  $L := L(T_I) \sim \log |I|$  denotes the depth of  $T_I$ . The following lemma states that the distance of the product  $Q_1 \cdot \dots \cdot Q_L$  to a unitary matrix is of the order  $\varepsilon \log L$  if the rounding precision is increased from level to level as  $\varepsilon/\ell$ .

**Lemma 2.34.** *Let  $0 < \varepsilon < 1$ . Assume that  $Q_\ell, \ell = 1, \dots, L$ , are approximately unitary in the sense that  $(Q_\ell + E_\ell)^H(Q_\ell + E_\ell) = I$  for some matrices  $E_\ell \in \mathbb{C}^{I \times I}$  satisfying  $\|E_\ell\|_2 < \varepsilon/\ell$ . Then also the product  $Q_L \cdot \dots \cdot Q_1$  is approximately unitary; i.e.,*

$$(Q_L \cdot \dots \cdot Q_1 + E)^H(Q_L \cdot \dots \cdot Q_1 + E) = I$$

for some  $E \in \mathbb{C}^{I \times I}$  satisfying  $\|E\|_2 \sim \varepsilon \log L$ .

*Proof.* The assertion is proved by induction over  $L$ . The case  $L = 1$  is trivially true. Assume that for  $Q := Q_L \cdot \dots \cdot Q_1$  it holds that

$$(Q + E)^H(Q + E) = I$$

for some  $E \in \mathbb{C}^{I \times I}$  satisfying  $\|E\|_2 \leq \varepsilon \left( \sum_{\ell=1}^L 1/\ell \right) \prod_{\ell=2}^L (1 + \frac{\varepsilon}{\ell})$ . Observe that

$$[Q_{L+1}Q + F]^H[Q_{L+1}Q + F] = [(Q_{L+1} + E_{L+1})(Q + E)]^H(Q_{L+1} + E_{L+1})(Q + E) = I,$$

where  $F := (Q_{L+1} + E_{L+1})E + E_{L+1}(Q + E) - E_{L+1}E$ . The norm of  $F$  can be estimated as

$$\begin{aligned} \|F\|_2 &\leq \|E\|_2 + \|E_{L+1}\|_2 + \|E_{L+1}\|_2\|E\|_2 \\ &\leq \varepsilon \left( \sum_{\ell=1}^L 1/\ell \right) \prod_{\ell=2}^L (1 + \frac{\varepsilon}{\ell}) + \frac{\varepsilon}{L+1} + \frac{\varepsilon^2}{L+1} \left( \sum_{\ell=1}^L 1/\ell \right) \prod_{\ell=2}^L (1 + \frac{\varepsilon}{\ell}) \\ &\leq \varepsilon \left( \sum_{\ell=1}^{L+1} 1/\ell \right) \prod_{\ell=2}^L (1 + \frac{\varepsilon}{\ell}) + \frac{\varepsilon^2}{L+1} \left( \sum_{\ell=1}^{L+1} 1/\ell \right) \prod_{\ell=2}^L (1 + \frac{\varepsilon}{\ell}) \\ &= \varepsilon \left( \sum_{\ell=1}^{L+1} 1/\ell \right) \prod_{\ell=2}^{L+1} (1 + \frac{\varepsilon}{\ell}). \end{aligned}$$

The assertion follows from  $\prod_{\ell=2}^L (1 + \frac{\varepsilon}{\ell}) \leq \exp(\varepsilon)$  and  $\sum_{\ell=1}^L 1/\ell \sim \log L$ .  $\square$

Assume a logarithmic dependence of the required rank  $k$  on the prescribed accuracy. Then increasing the accuracy with the level  $\ell$  does not significantly increase the rank due to  $k \sim |\log \varepsilon/\ell| \sim |\log \varepsilon| + \log \ell \sim |\log \varepsilon| + \log \log |I|$ .

## 2.11 $\mathcal{H}^2$ -Matrices and Fast Multipole Methods

The low-rank blocks of  $\mathcal{H}$ -matrices can be chosen independently from each other. By fixing a common basis for the column and the row vectors it is possible to further reduce the complexity of data-sparse approximations.

**Definition 2.35.** Let  $T_I$  be a cluster tree and let  $k_t \in \mathbb{N}$ ,  $t \in T_I$ , be given. A family of matrices  $V(t) \in \mathbb{C}^{t \times k_t}$  is called **cluster basis** for the cluster tree  $T_I$  and the rank distribution  $(k_t)_{t \in T_I}$ .

Let cluster bases  $\mathcal{U} := \{U(t), t \in T_I\}$  and  $\mathcal{V} := \{V(s), s \in T_J\}$  with associated rank distributions  $(k_t^{\mathcal{U}})_{t \in T_I}$  and  $(k_s^{\mathcal{V}})_{s \in T_J}$  be given. The following set of uniform  $\mathcal{H}$ -matrices (see [127]) is a subset of the set of  $\mathcal{H}$ -matrices.

**Definition 2.36.** Let  $T_{I \times J}$  be a block cluster tree generated from the cluster trees  $T_I$  and  $T_J$ . Furthermore, let  $\mathcal{U}$  and  $\mathcal{V}$  be cluster bases for  $T_I$  and  $T_J$ . A matrix  $A \in \mathbb{C}^{I \times J}$  satisfying

$$A_{ts} = U(t)S(t,s)V(s)^H \quad \text{for all admissible } t \times s \in \mathcal{L}(T_{I \times J}) \quad (2.20)$$

and some  $S(t,s) \in \mathbb{C}^{k_t^{\mathcal{U}} \times k_s^{\mathcal{V}}}$  is called **uniform hierarchical matrix** for  $\mathcal{U}$  and  $\mathcal{V}$ .

Condition (2.20) means that  $A_{ts}$  is in the subspace

$$\text{span}\{U_i(t)V_j(s)^H, i = 1, \dots, k_t^{\mathcal{U}}, j = 1, \dots, k_s^{\mathcal{V}}\},$$

where  $U_i$  and  $V_j$  denote the  $i$ th and the  $j$ th column of  $U$  and  $V$ , respectively. In contrast to  $\mathcal{H}$ -matrices, uniform  $\mathcal{H}$ -matrices of *common* cluster bases form a linear subspace of  $\mathbb{C}^{I \times J}$ , which avoids rounding sums. Note that this subspace property would also hold under the weaker condition that each of the two cluster bases depends on  $t$  and  $s$ , i.e.,

$$A_{ts} = U(t,s)S(t,s)V(t,s)^H$$

instead of (2.20).

Let  $k = \max\{k_t^{\mathcal{U}}, k_s^{\mathcal{V}}; t \in T_I, s \in T_J\}$ . If many uniform  $\mathcal{H}$ -matrices of common cluster bases  $\mathcal{U}$  and  $\mathcal{V}$  are to be stored,  $\mathcal{U}$  and  $\mathcal{V}$  should be stored separately from the coefficient matrices  $S(t,s)$ . The storage requirements of the latter are of the order  $k \min\{|I|, |J|\}$ , which due to

$$\sum_{t \in T_I} \sum_{s \in T_J} \min\{k^2, |t|^2\} \leq c_{\text{sp}} \sum_{t \in T_I} \min\{k^2, |t|^2\}$$

can be seen from (1.15). Here, we have assumed that for clusters  $t, s$  from the same level it holds that  $|t| \approx |s|$ .

However, storing the cluster bases still requires  $k[|I|L(T_I) + |J|L(T_J)]$  units of storage due to Lemma 1.21. This situation can be improved by introducing a second hierarchy. The following space of  $\mathcal{H}^2$ -matrices (see [136]) consists of uniform  $\mathcal{H}$ -matrices of given cluster bases  $\mathcal{U}$  and  $\mathcal{V}$  which are nested.

**Definition 2.37.** A cluster basis  $\mathcal{V}$  is called **nested** if for each  $t \in T \setminus \mathcal{L}(T)$  there are transfer matrices  $B_{t't} \in \mathbb{C}^{k_{t'} \times k_t}$  such that

$$(V(t))_{t'} = V(t')B_{t't} \quad \text{for all } t' \in S(t).$$

Storing a nested cluster basis requires storing  $V(t)$  for all leaf clusters  $t \in \mathcal{L}(T)$  and the transfer matrices  $B_{t't}$ ,  $t' \in S(t)$ , for all  $t \in T \setminus \mathcal{L}(T)$ , which can be done with order  $k|I|$  units of storage due to (1.15); cf. [129].

**Definition 2.38.** An  $\mathcal{H}^2$ -**matrix** is a uniform  $\mathcal{H}$ -matrix with nested cluster bases.

While in [136] the Taylor expansion is used to show existence of  $\mathcal{H}^2$ -matrix approximants, in [103] a practical procedure for their construction is proposed which is based on polynomial interpolation. Since polynomials do not provide an optimal approximation system, algebraic recompression techniques [44, 41] can be used to improve the approximant. The complexity  $k|I|$  can also be achieved by the method presented in Sect. 3.5, which uses the conceptionally easier uniform  $\mathcal{H}$ -matrices.

### 2.11.0.2 Matrix-Vector Multiplication

The matrix-vector multiplication  $y := y + Ax$  of an  $\mathcal{H}^2$ -matrix  $A$  by a vector  $x \in \mathbb{C}^J$  can be done by the following three-phase algorithm; cf. [136].

1. *Forward transform*

In this first phase, transformed vectors  $\hat{x}(s) := V(s)^H x_s$  are computed for all  $s \in T_J$ . Exploiting the nestedness of the cluster basis  $\mathcal{V}$ , one has the following recursive relation

$$\hat{x}(s) = V(s)^H x_s = \sum_{s' \in S(s)} B_{s's}^H V(s')^H x_{s'} = \sum_{s' \in S(s)} B_{s's}^H \hat{x}(s'),$$

which has to be applied starting from the leaf vectors  $\hat{x}(s) = V(s)^H x_s$ ,  $s \in \mathcal{L}(T_J)$ .

2. *Far field interaction*

In the second phase the products  $S(t, s)\hat{x}(s)$  are computed and summed up over all clusters in  $R_t := \{s \in T_J : t \times s \in P \text{ is admissible}\}$ :

$$\hat{y}(t) := \sum_{s \in R_t} S(t, s)\hat{x}(s), \quad t \in T_I.$$

3. *Backward transform*

In this third phase, the vectors  $\hat{y}(t)$  are transformed to the target vector  $y$ . The nestedness of the cluster basis  $\mathcal{U}$  provides the following recursion which descends the cluster tree  $T_I$

- (a) Compute  $\hat{y}(t') := \hat{y}(t') + B_{t't}\hat{y}(t)$  for all  $t' \in S(t)$ ;
- (b) Compute  $y_t := y_t + U(t)\hat{y}(t)$  for all leaf clusters  $t \in \mathcal{L}(T_I)$ .

#### 4. Near field interaction

Compute  $y_t := y_t + A_{ts}x_s$  for all non-admissible blocks  $t \times s \in P$ .

The previous matrix-vector multiplication is an algebraic generalization of the **fast multipole method**; see [121] and the improved version [122]. The number of operations is of the order of the number of involved matrix entries and hence  $k[|I| + |J|]$ .  $\mathcal{H}^2$ -matrices will therefore improve the asymptotic complexity of  $\mathcal{H}$ -matrices by a single logarithmic factor due to the nestedness of the cluster bases. If **variable order techniques** are used, i.e., the rank

$$k(\ell) = [\alpha(\mathcal{L}(T_I) - \ell) + \beta]^\gamma$$

with parameters  $\alpha, \beta$ , and  $\gamma$  is chosen depending on the level  $\ell$  of a block, then one can guarantee linear complexity; cf. [136, 227, 184]. Variable order approximations, however, are feasible only in very special situations.

In addition to the matrix-vector multiplication,  $\mathcal{H}^2$ -matrices admit matrix operations such as matrix addition and matrix multiplication with different cluster bases; see [42] for addition and multiplication algorithms of complexity  $k^3|I|$ . For these operations, however, the problem arises that the cluster bases required to hold the results of addition and multiplication differ from the bases of the input matrices and are usually unknown.

## 2.12 Using Hierarchical Matrices for Preconditioning

When investigating the complexity of algorithms for the solution of linear systems arising from the discretization of a continuous problem, one considers a sequence of systems

$$A_n x_n = b_n, \quad n \rightarrow \infty, \quad (2.21)$$

where each  $A_n \in \mathbb{C}^{n \times n}$  is invertible. However, for the sake of readability the index  $n$  is usually dropped whenever this dependency is obvious from the context. For the solution of (2.21) either direct or iterative solvers can be used. The complexity of direct solvers such as Gaussian elimination applied to fully populated linear systems is of cubic order and hence prohibitively large. There are improved variants (cf. [78, 4, 230]) of Gaussian elimination if  $A$  is sparse. Due to *fill-in*, these methods are efficient only in two spatial dimensions. The presented hierarchical  $LU$  decomposition from Sect. 2.9 could in principle be used to compute an approximate  $LU$  decomposition with almost linear complexity.  $\mathcal{H}$ -matrices, however, can be multiplied by a vector with complexity  $kn \log n$  and a much smaller (hidden) constant. The iterative Krylov subspace methods (cf. [223]) such as the **conjugate gradient method** (CG) [145] or the **method of generalized minimal residuals** (GMRes) [224], which the matrix  $A$  enters only through the matrix-vector product, will hence be faster provided that the number of iterations is small enough.

It is well known that the convergence of Krylov subspace methods is determined by spectral properties such as the distribution of eigenvalues if  $A$  is normal or the numerical range in the general case. Depending on the mapping properties of the underlying differential or integral operator  $\mathcal{A}$ , its discretization  $A$  and hence also its  $\mathcal{H}$ -matrix approximant  $A_{\mathcal{H}}$  may be ill-conditioned. In addition, a large condition number can result from the coefficients of the operator or the discretization of the geometry even for small  $n$ ; see Sect. 3.6. Therefore, if (2.21) is to be solved iteratively, one has to incorporate a preconditioner.

A **left preconditioner** is a regular and easily invertible matrix  $C$  such that  $C \approx A_{\mathcal{H}}^{-1}$  in the sense that the distribution of eigenvalues of  $CA_{\mathcal{H}}$  leads to an improved convergence of the respective iteration scheme, where instead of (2.21) one solves the equivalent linear system

$$CA_{\mathcal{H}}x = Cb.$$

The convergence rate of Krylov subspace methods in the Hermitian case is determined by the spectral condition number of  $CA_{\mathcal{H}}$ . Hence,  $C$  has to be chosen such that  $\text{cond}_2(CA_{\mathcal{H}}) \sim 1$ . If  $A_{\mathcal{H}}$  is non-Hermitian, the aim of preconditioning is to obtain a spectrum of  $CA_{\mathcal{H}}$  which is clustered away from the origin. When Krylov subspace methods are used, it is not necessary to form the preconditioned matrix  $CA_{\mathcal{H}}$ . Instead, vectors should be multiplied by  $A_{\mathcal{H}}$  and  $C$  consecutively. If  $C$  is used as a **right preconditioner**, (2.21) is replaced by

$$A_{\mathcal{H}}C\tilde{x} = b.$$

In the latter case, the solution  $x$  can be obtained as  $x = C\tilde{x}$ . In this section only right preconditioners are considered. Left preconditioners can be constructed analogously.

Since  $\mathcal{H}$ -matrices provide efficient approximations to the inverse and to the  $LU$  decomposition, they are particularly suited for preconditioning. Although the results of this section are proved for the approximate inverse, they are obviously also valid if the approximate  $LU$  decomposition  $A_{\mathcal{H}} \approx L_{\mathcal{H}}U_{\mathcal{H}}$  is used as  $C = (L_{\mathcal{H}}U_{\mathcal{H}})^{-1}$ . The aim of this section is to establish a relation between the condition number of  $A_{\mathcal{H}}C$  and the approximation accuracy  $\varepsilon$  of the inverse in the Hermitian case. For non-Hermitian coefficient matrices we will present lower bounds for the distance of a cluster of eigenvalues of  $A_{\mathcal{H}}C$  and the origin; cf. [22]. This relation will be used to find out the required size of  $\varepsilon$ . It will be seen that a low-accuracy approximate inverse of  $A_{\mathcal{H}}$  is sufficient to guarantee a bounded number of preconditioned iterations of appropriate iterative schemes. In addition, the derived condition will guarantee required properties of the preconditioner such as invertibility and positivity. Moreover, the number of iterations will depend neither on the operator nor on the computational domain but only on the accuracy  $\varepsilon$ . Numerical experiments in Sect. 3.6 and Sect. 4.4 will demonstrate the effectiveness of the preconditioner when it is applied to fully populated matrices arising from the discretization of integral operators and to sparse discretizations of partial differential operators.

In the first part of this section the case of Hermitian positive definite coefficient matrices  $A_{\mathcal{H}}$  is treated, the second part concentrates on the non-Hermitian case.

### 2.12.1 Hermitian Positive Definite Coefficient Matrices

Depending on the operator  $\mathcal{A}$ , the approximation  $A_{\mathcal{H}}$  to the discrete operator  $A \in \mathbb{R}^{n \times n}$  is often Hermitian positive definite. If (2.21) is to be solved iteratively, the **preconditioned conjugate gradient method** (PCG) is the method of choice. Its convergence rate and hence the number of iterations required to obtain a prescribed accuracy of the solution is determined by the distribution of eigenvalues of  $A_{\mathcal{H}}$ ; see Theorem 2.40.

Typically, the condition number of  $A$  grows for an increasing number of unknowns  $n$ . The aim of this section is to present preconditioners  $C$  such that the number of iterations for the preconditioned coefficient matrix  $A_{\mathcal{H}}C$  is bounded by a constant. A bounded number of iterations, in turn, is guaranteed by an asymptotically well-conditioned matrix  $A_{\mathcal{H}}C$ .

**Definition 2.39.** Let  $\{A_n\}_{n \in \mathbb{N}}$  be a sequence of Hermitian matrices. Assume that there is a constant  $c > 0$  such that

$$\text{cond}_2(A_n) \leq c \quad (2.22)$$

for all  $n \in \mathbb{N}$ . Then  $\{A_n\}_{n \in \mathbb{N}}$  is said to be **asymptotically well-conditioned**.

The following theorem (cf. [9]) describes the convergence of the conjugate gradient method. The estimate is formulated in terms of the norm  $\|x\|_{A_{\mathcal{H}}} := \|A_{\mathcal{H}}x\|_2$ .

**Theorem 2.40.** Let the spectrum of  $A_{\mathcal{H}}C$  be decomposed in the following way:

$$\sigma(A_{\mathcal{H}}C) = \{\lambda'_i, i = 1, \dots, p\} \cup \Lambda \cup \{\lambda''_i, i = 1, \dots, q\}, \quad \Lambda \subset [a, b].$$

Then for the error  $x_k - x$ ,  $k = 0, \dots, n-1$ , of the iterate  $x_k$  of PCG it holds that

$$\|x_k - x\|_{A_{\mathcal{H}}} \leq 2(\text{cond}_2(A_{\mathcal{H}}C) + 1)^p \left( \frac{\sqrt{b/a} - 1}{\sqrt{b/a} + 1} \right)^{k-p-q} \|x_0 - x\|_{A_{\mathcal{H}}}.$$

Hence, if  $A_{\mathcal{H}}C$  is asymptotically well-conditioned, we may choose  $\Lambda = \sigma(A_{\mathcal{H}}C)$ . In this case, PCG converges linearly, and the number of iterations depends only on the condition number of  $A_{\mathcal{H}}C$  and not on  $n$ .

Although asymptotically well-conditioned coefficient matrices lead to a bounded number of iterations, this number might still be large since the constant in (2.22) usually depends on the coefficients of the underlying operator  $\mathcal{A}$ , the geometry, and its discretization. This influence might be even more severe than the dependence on  $n$ ; see the numerical experiments in Sect. 3.6 and Sect. 4.4. As we shall see in the next lemma, the condition

$$\|I - A_{\mathcal{H}}C\|_2 \leq \varepsilon < 1, \quad (2.23)$$

in which  $\varepsilon$  does not depend on  $n$ , leads to an asymptotically well-conditioned matrix  $A_{\mathcal{H}}C$ . Spectral equivalence of two matrices  $A$  and  $B$  does not require  $A$  to

approximate  $B$ . The matrix  $2B$ , for instance, has the same preconditioning effect as  $B$ . However, if they approximate in a certain sense, any condition number in the neighborhood of 1 can be achieved by decreasing the approximation error. Especially, this allows to guarantee problem-independent convergence rates, whereas non-approximating preconditioners usually are only able to guarantee the boundedness of the condition number.

**Lemma 2.41.** *Assume that (2.23) holds. Then*

$$\text{cond}_2(A_{\mathcal{H}}C) = \|A_{\mathcal{H}}C\|_2 \|(A_{\mathcal{H}}C)^{-1}\|_2 \leq \frac{1+\varepsilon}{1-\varepsilon}. \quad (2.24)$$

*Proof.* The assertion follows from the triangle inequality

$$\|A_{\mathcal{H}}C\|_2 \leq \|I\|_2 + \|I - A_{\mathcal{H}}C\|_2 \leq 1 + \varepsilon$$

and from the Neumann series

$$\|(A_{\mathcal{H}}C)^{-1}\|_2 \leq \sum_{k=0}^{\infty} \|I - A_{\mathcal{H}}C\|_2^k = \frac{1}{1-\varepsilon}.$$

□

Note that (2.24) provides an explicit bound on the condition number. The choice  $\varepsilon = 0.5$ , for instance, guarantees that  $\text{cond}_2(A_{\mathcal{H}}C) \leq 3$ . In addition, condition (2.23) guarantees that  $C$  is non-singular. To see this, let  $\lambda$  be an eigenvalue of  $A_{\mathcal{H}}C$  with associated eigenvector  $z$ ,  $\|z\|_2 = 1$ , then

$$|1 - \lambda| = \|(I - A_{\mathcal{H}}C)z\|_2 \leq \|I - A_{\mathcal{H}}C\|_2 \leq \varepsilon < 1. \quad (2.25)$$

Hence, with  $A_{\mathcal{H}}C$  also  $C$  is non-singular. In order to be able to apply PCG,  $C$  additionally needs to be Hermitian positive definite. It is interesting to see that this is already guaranteed by condition (2.23).

**Lemma 2.42.** *Assume that  $A_{\mathcal{H}}$  is Hermitian positive definite. Then any Hermitian matrix  $C$  satisfying (2.23) is positive definite, too.*

*Proof.* According to the assumptions, the square root  $A_{\mathcal{H}}^{1/2}$  of  $A_{\mathcal{H}}$  is defined. Since  $A_{\mathcal{H}}C$  is similar to the Hermitian matrix  $A_{\mathcal{H}}^{1/2}CA_{\mathcal{H}}^{1/2}$ , the eigenvalues of  $A_{\mathcal{H}}C$  are real. Moreover, for the smallest eigenvalue of  $A_{\mathcal{H}}^{1/2}CA_{\mathcal{H}}^{1/2}$  it follows from (2.25) that

$$\lambda_{\min}(A_{\mathcal{H}}^{1/2}CA_{\mathcal{H}}^{1/2}) = \lambda_{\min}(A_{\mathcal{H}}C) \geq 1 - \varepsilon.$$

Let  $x \neq 0$  and  $y = A_{\mathcal{H}}^{-1/2}x$ . Then  $y \neq 0$  and we have

$$x^H C x = y^H A_{\mathcal{H}}^{1/2} C A_{\mathcal{H}}^{1/2} y \geq (1 - \varepsilon) \|y\|_2^2 > 0,$$

which proves that  $C$  is positive definite. □

From the approximation by  $\mathcal{H}$ -matrices usually error estimates of the form

$$\|A_{\mathcal{H}} - C^{-1}\|_2 \leq \varepsilon \|A_{\mathcal{H}}\|_2 \quad \text{or} \quad \|A_{\mathcal{H}}^{-1} - C\|_2 \leq \varepsilon \|A_{\mathcal{H}}^{-1}\|_2 \quad (2.26)$$

instead of (2.23) are satisfied. In this case, the following lemma describes a sufficient condition on  $\varepsilon$ .

**Lemma 2.43.** *Assume that (2.26) holds with  $\varepsilon > 0$  such that  $\varepsilon' := \varepsilon \operatorname{cond}_2(A_{\mathcal{H}}) < 1$ . Then*

$$\operatorname{cond}_2(A_{\mathcal{H}}C) \leq \frac{1 + \varepsilon'}{1 - \varepsilon'}.$$

*Proof.* Assume first that  $\|A_{\mathcal{H}} - C^{-1}\|_2 \leq \varepsilon \|A_{\mathcal{H}}\|_2$ . Since

$$\|I - (A_{\mathcal{H}}C)^{-1}\|_2 = \|(A_{\mathcal{H}} - C^{-1})A_{\mathcal{H}}^{-1}\|_2 \leq \varepsilon \|A_{\mathcal{H}}\|_2 \|A_{\mathcal{H}}^{-1}\|_2 = \varepsilon \operatorname{cond}_2(A_{\mathcal{H}}),$$

one can apply the estimates of the proof of Lemma 2.41 with  $A_{\mathcal{H}}C$  replaced by  $(A_{\mathcal{H}}C)^{-1}$ .

If  $\|A_{\mathcal{H}}^{-1} - C\|_2 \leq \varepsilon \|A_{\mathcal{H}}^{-1}\|_2$ , then

$$\|I - A_{\mathcal{H}}C\|_2 = \|A_{\mathcal{H}}(A_{\mathcal{H}}^{-1} - C)\|_2 \leq \varepsilon \|A_{\mathcal{H}}\|_2 \|A_{\mathcal{H}}^{-1}\|_2 = \varepsilon \operatorname{cond}_2(A_{\mathcal{H}})$$

gives the assertion.  $\square$

The stronger condition  $\varepsilon \operatorname{cond}_2(A_{\mathcal{H}}) < 1$  implies that  $\varepsilon$  has to go to zero if  $A_{\mathcal{H}}$  is not well-conditioned. This, however, will not destroy the almost linear complexity since it will be seen that the complexity of the  $\mathcal{H}$ -matrix approximation depends logarithmically on the accuracy  $\varepsilon$ .

### 2.12.1.1 Clusters of Eigenvalues

From Theorem 2.40 it can also be seen that few small or large eigenvalues  $\lambda_i'$  and  $\lambda_i''$  do not affect the rate of convergence. Therefore, the distribution of eigenvalues within the spectrum is even more important for the rate of convergence than the condition number, which depends only on the minimum and maximum eigenvalue of  $A_{\mathcal{H}}C$ . A faster convergence of PCG can be obtained by a condition on the distribution neglecting the size of the interval.

**Definition 2.44.** By  $\gamma_A(\varepsilon)$  we denote the number of eigenvalues of  $A \in \mathbb{R}^{n \times n}$  lying outside a disc of radius  $\varepsilon > 0$  centered at the origin. The eigenvalues of a sequence of matrices  $\{A_n\}_{n \in \mathbb{N}}$  are said to have a **cluster** at 0 if  $\gamma_{A_n}(\varepsilon)$  is bounded independently of  $n$ . The eigenvalues of  $\{A_n\}_{n \in \mathbb{N}}$  are said to have a cluster at  $z \in \mathbb{C}$  if  $\{A_n - zI_n\}_{n \in \mathbb{N}}$  has a cluster at 0.

If  $A_{\mathcal{H}}$  and  $C$  are Hermitian positive definite and  $A_{\mathcal{H}}C$  has a cluster at 1, then PCG converges super-linearly; i.e., for all sufficiently large  $n$  the residual in the  $k$ th step is bounded by  $cq^k$  for all  $0 < q < 1$ . Why this super-linear convergence happens is explained in [9].



The following theorem states that for the existence of eigenvalue clusters the approximation  $C$  of  $A_{\mathcal{H}}^{-1}$  does not have to be too accurate.

**Theorem 2.45.** *Let  $\{A_n\}_{n \in \mathbb{N}} \subset \mathbb{R}^{n \times n}$  be a bounded sequence, i.e.,  $\|A_n\|_F \leq c$ , where  $c$  does not depend on  $n$ . Then both the singular values and the eigenvalues of  $\{A_n\}_{n \in \mathbb{N}}$  cluster at 0.*

*Proof.* By  $v_A(\varepsilon)$  we denote the number of singular values  $\sigma_i(A)$ ,  $i = 1, \dots, n$ , of  $A$  lying outside a disc of radius  $\varepsilon > 0$  centered at the origin. Assume that  $v_A(\varepsilon) > c^2/\varepsilon^2$ . Then

$$c^2 < v_A(\varepsilon)\varepsilon^2 \leq \sum_{i=1}^n \sigma_i^2(A) = \|A\|_F^2,$$

which gives the contradiction. Hence,  $v_A(\varepsilon) \leq c^2/\varepsilon^2$ .

In order to prove the same property for the eigenvalues, let  $A = QTQ^H$  be Schur's form with unitary  $Q$  and upper triangular matrix  $T$  with the eigenvalues of  $A$  on its diagonal. The assertion follows from

$$\|A\|_F^2 = \|T\|_F^2 \geq \sum_{i=1}^n |\lambda_i(A)|^2$$

and the same arguments as above.  $\square$

Therefore, if for  $C$  it holds that

$$\|I - A_{\mathcal{H}}C\|_F \leq c$$

with a constant  $c > 0$  which does not depend on  $n$ , then the eigenvalues of  $A_{\mathcal{H}}C$  cluster at 1 and PCG converges super-linearly. Note that in order to guarantee that  $C$  is positive definite, we can employ the stabilized rounded addition from Sect. 2.5.1 during the computations without any further assumption on  $c$ .

### 2.12.2 Non-Hermitian Coefficient Matrices

If  $\mathcal{A}$  is not self-adjoint, then  $A_{\mathcal{H}}$  cannot be expected to be Hermitian. In this case, not the spectral condition number of the coefficient matrix but the distance of a cluster of eigenvalues to the origin usually determines the convergence rate of appropriate Krylov subspace methods such as GMRes, BiCGStab, and MinRes. Nevertheless, a low-accuracy approximate inverse will be sufficient to guarantee a bounded number of iterations.

For the convergence of GMRes, for instance, the **numerical range**

$$\mathcal{F}(A_{\mathcal{H}}C) := \{x^H A_{\mathcal{H}}C x : x \in \mathbb{C}^n, \|x\|_2 = 1\}$$

of  $A_{\mathcal{H}}C$  is of particular importance. It is known (see [117]) that for the  $k$ th iterate  $x_k$  of GMRes applied to  $Ax = b$  it holds that

$$\|b - A_{\mathcal{H}}x_k\|_2 \leq 2 \left( \frac{r}{|z|} \right)^k \|b\|_2$$

provided that  $\mathcal{F}(A_{\mathcal{H}}C) \subset B_r(z)$ , where  $B_r(z)$  denotes the closed disc around  $z$  with radius  $r$ . A similar behavior can be observed for other non-Hermitian Krylov subspace methods. Condition (2.23) implies that  $\mathcal{F}(A_{\mathcal{H}}C) \subset B_\varepsilon(1)$ , which follows from

$$|x^H A_{\mathcal{H}}C x - 1| = |x^H (A_{\mathcal{H}}C - I)x| \leq \|I - A_{\mathcal{H}}C\|_2 \leq \varepsilon \quad \text{for all } x \in \mathbb{C}^n, \|x\|_2 = 1.$$

Therefore, (2.23) also leads to a problem-independent convergence

$$\|b - A_{\mathcal{H}}x_k\|_2 \leq 2\varepsilon^k \|b\|_2 \tag{2.27}$$

of GMRes.

In the following chapters these results will be used when solving problems arising from the discretization of integral operators and boundary value problems.

<http://www.springer.com/978-3-540-77146-3>

Hierarchical Matrices

A Means to Efficiently Solve Elliptic Boundary Value Problems

Bebendorf, M.

2008, XVI, 296 p., Softcover

ISBN: 978-3-540-77146-3