Eigenvalue Algorithms for Symmetric Hierarchical Matrices

Thomas Mach

Eigenvalue Algorithms for Symmetric Hierarchical Matrices

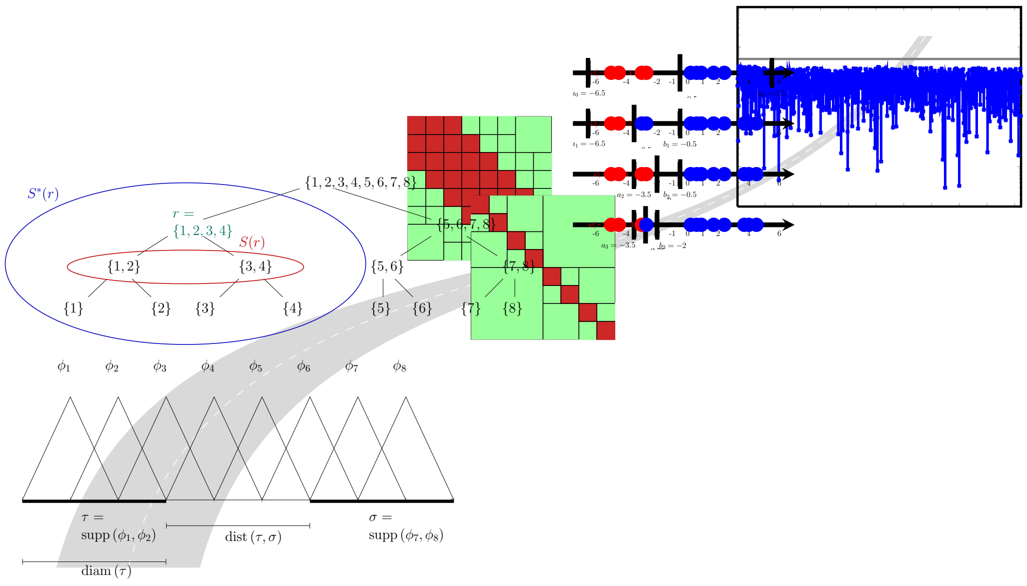
Dissertation

submitted to **Department of Mathematics**

at **Chemnitz University of Technology**

in accordance with the requirements for the degree

Dr. rer. nat.



presented by: Dipl.-Math. techn. Thomas Mach

Advisor: Prof. Dr. Peter Benner

Reviewer: Prof. Dr. Steffen Börm

March 13, 2012

ACKNOWLEDGMENTS

Personal Thanks. I thank Peter Benner for giving me the time and the freedom to follow several ideas and for helping me with new ideas. I would also like to thank my colleagues in Peter Benner's groups in Chemnitz and Magdeburg. All of them helped me a lot during the last years. The following four will be highlighted, since they are most important for me. I would like to thank Jens Saak for explaining me countless things and for providing his LaTeX framework that I used for this document. I stressed our compute cluster specialist, Martin Köhler, a lot with questions and request. I thank him for his help. Further, I am greatly indebted to Ulrike Baur, who explained me many details concerning hierarchical matrices. I want to thank Martin Stoll for pushing me forward during the final year.

I also want to thank Jessica Gördes and Steffen Börm (CAU Kiel) for discussing eigenvalue algorithms for $\mathcal{H}_\ell$-matrices, Marc Van Barel (KU Leuven) for pointing out the relations to semiseparable matrices, and Lars Grasedyck (RWTH Aachen) for a discussion on QR decompositions and the QR-like algorithms for $\mathcal{H}$-matrices.

I am indebted to Tom Barker for carefully proof reading this thesis.

Finally this work would not be possible without adequate recreation by (extensive) sports activities. Therefore, I thank my friends Benjamin Schulz, Jens Lang, and Tobias Breiten for cycling and jogging with me.

Last but not least I thank my sister for proof reading and my parents, who made all of this possible in the first place.

Financial Support. This research was supported by a grant from the Free State of Saxony (Sächsisches Landesstipendium) for two years.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF ACRONYMS

BEM	boundary element method
BLAS	basic linear algebra subroutines
dpss	diagonal plus semiseparable matrix . . . . Definition 2.5.2
EVP	eigenvalue problem . . . . Definition 2.1.6
FDM	finite difference method
FEM	finite element method
flop	one floating point operation of the form $y = x + y$ or $y = \alpha y,\ x, y, \alpha \in \mathbb{R}^1$
flops	plural of flop
Fortran	procedural programming language, e.g., used for BLAS
$\mathcal{H}$	hierarchical
$\mathcal{H}$Lib	libary for hierarchical matrices . . . . [65]
LAPACK	linear algebra package . . . . [2]
MATLAB®	software from The MathWorks Inc. for numerical computations
MPI	Max Planck Institute
nnz	number of non-zero entries of a sparse matrix
OpenMP	open multi-processing . . . . [84]
Open MPI	open message passing interface . . . . [97]
Otto	compute cluster at the MPI Magdeburg . . . . Section 2.7
($\mathcal{H}$-)PINVIT	(hierarchical) preconditioned inverse iteration . . . Section 6.2
RAM	random-access memory
SVD	singular value decomposition . . . . Equation (2.5)
Xeon®	processor series from Intel®

LIST OF SYMBOLS

Sets and Spaces

$B_c(x)$	ball around x, $B_c(x) := \{y \mid \|x-y\| < c\}$
$I, J \subset \mathbb{N}$	sets of indices, e.g., $I = \{1, \dots, n\}$
$\|I\|$	cardinality of I
$\mathcal{H}(T_{I\times I}, k, \mathcal{A})$	set of hierarchical matrices . . . . . Definition 2.3.6
$\mathcal{H}_\ell(k)$	set of $\mathcal{H}_\ell$-matrices with rank k and depth ℓ . . . . . Definition 2.3.12
$\mathrm{HSS}(k)$	set of hierarchically semiseparable matrices . . . . . Definition 2.5.7
$\mathcal{L}(T)$	set of leaves of T . . . . . Equation (2.11)
$\mathcal{L}^{(i)}(T)$	set of leaves of T on level i . . . . . Equation (2.12)
$\mathcal{L}^{+}(T_{I\times J})$	set of admissible leaves of $T_{I\times J}$ . . . . . Equation (2.14)
$\mathcal{L}^{-}(T_{I\times J})$	set of inadmissible leaves of $T_{I\times J}$ . . . . . Equation (2.15)
$\mathcal{M}_{k,\tau}$	set of matrices with a block $M_{I\setminus\tau\times\tau}$ of rank k . . . . . Equation (2.19)
$\mathbb{N}$	set of positive integers
$\mathcal{O}(g(n))$	set of functions with $\mathcal{O}(n) = \left\{ f(n) \middle\| \lim\limits_{n\to\infty} \left\|\frac{f(n)}{g(n)}\right\| < \infty \right\}$
$o(g(n))$	set of functions with $\mathcal{O}(n) = \left\{ f(n) \middle\| \lim\limits_{n\to\infty} \left\|\frac{f(n)}{g(n)}\right\| = 0 \right\}$
$\mathbb{R}$	field of real numbers
$\mathbb{R}^n, \mathbb{R}^I$	real vectors of length n, $\|I\|$ . . . . . Definition 2.1.1
$\mathbb{R}^{n\times m}, \mathbb{R}^{I\times J}$	real matrices of size $n \times m$, $\|I\| \times \|J\|$ . . . . . Definition 2.1.2
$S(r)$	sons of $r \in T$ . . . . . Definition 2.3.2
$S^*(r)$	descendants of $r \in T$ . . . . . Equation (2.8)
T	tree $T = (V, E)$, connected graph without cycles
T_I	hierarchical tree . . . . . Definition 2.3.2
$T_{I\times I}$	hierarchical block tree . . . . . Definition 2.3.5
$T^{(i)}$	level i of tree T . . . . . Equation (2.9)

Matrices

A, B	(thin) rectangular matrices, typically $A, B \in \mathbb{R}^{\cdot\times k}$
B^T	transpose of B
$\mathrm{diag}\,(d)$	diagonal matrix with diagonal $d \in \mathbb{R}^n$
$\mathrm{diag}\,(M)$	diagonal of the matrix $M \in \mathbb{R}^{n\times n}$

I_n, I	identity matrix of size $n \times n$ resp. of suitable size
e_i	i-th column of the identity matrix I
$\kappa(M)$	condition number of M .... Equation (2.3)
$\kappa_{\text{EVP}}(M)$	condition number of the eigenvalue problem of M ... Lemma 2.1.12
$\Lambda(M)$	spectrum of matrix M .... Definition 2.1.6
$\lambda_i(M), \lambda_i$	$\lambda_i \in \Lambda(M)$ is the i-th smallest eigenvalue .... Definition 2.1.6
M	symmetric matrix .... Definition 2.1.2
M_{ij}	entry (i, j) of M
$M_{k:l,m:n}$	submatrix of M containing the entries $M_{i,j}$ with row index $i \in \{k, \ldots, l\}$ and column index $j \in \{m, \ldots, n\}$
$M_{k,\cdot}, M_{\cdot,m}$	k-th row resp. m-th column of M
$M = M^T > 0$	M is symmetric positive definite
$\nu_M(\mu)$	number of eigenvalues of M smaller than μ .... Equation (5.1)
$\operatorname{rank}(M)$	rank of M .... Definition 2.1.3
$\Sigma(M)$	set of singular values resp. diagonal matrix with singular values in descending order on the diagonal .... Equation (2.5)
$\sigma_i(M), \sigma_i$	i-th largest singular value of M .... Equation (2.5)
$\operatorname{tr}(M)$	trace of M, $\operatorname{tr}(M) = \sum_{i=1}^n m_{ii}$ .... Equation (2.2)
$\operatorname{tril}(M)$	lower triangular part of M .... Equation (2.22)
$\operatorname{triu}(M)$	upper triangular part of M .... Equation (2.23)

Norms

$\lVert v \rVert_p$	the p-(vector-)norm of v for $1 \le p < \infty$, $v \in \mathbb{R}^n : \lVert v \rVert_p := (\sum_{i=1}^n v_i^p)^{\frac{1}{p}}$
$\lVert v \rVert_2$	the Euclidean norm $v \in \mathbb{R}^n : \lVert v \rVert_2 := \left(\sum_{i=1}^n v_i^2\right)^{1/2}$
$\lVert v \rVert_\infty$	the maximum norm , $\lVert v \rVert_\infty := \max_{i \in \{1,\ldots,n\}} \lvert v_i \rvert$
$\lVert M \rVert_{p,q}$	the (induced) (p, q)-matrix norm of $M \in \mathbb{R}^{n \times m}$, $1 \le p, q < \infty$: $\lVert M \rVert_{p,q} := \max_{0 \ne x \in \mathbb{R}^m} \lVert Mx \rVert_q / \lVert x \rVert_p$
$\lVert M \rVert_1$	column sum norm $\lVert M \rVert_1 := \max_j \sum_i \lvert M_{ij} \rvert$
$\lVert M \rVert_2$	spectral norm $\lVert M \rVert_2 := \lVert M \rVert_{2,2} = \max_j \sqrt{\lambda_j(M^T M)}$
$\lVert M \rVert_2^{\mathcal{H}}$	approx. spectral norm $\lVert M \rVert_2$ computed by power iteration
$\lVert M \rVert_\infty$	row sum norm $\lVert M \rVert_\infty := \max_i \sum_j \lvert M_{ij} \rvert$
$\lVert M \rVert_F$	Frobenius norm $\lVert M \rVert_F := \sqrt{\sum_{i,j=1}^n \lvert M_{ij} \rvert^2}$

Constants

C_{id}	idempotency constant .... Definition 2.3.8
C_{sp}	sparsity constant .... Definition 2.3.7
$\operatorname{depth}(T)$	depth of tree T .... Equation (2.10)
$n_{\min}$	minimal block size .... Definition 2.3.6

PUBLICATIONS

The main parts of this thesis have been published or are submitted for publication. Chapter 3 has been published in

> [11]: Peter Benner, Thomas Mach: On the QR decomposition of $\mathcal{H}$-matrices, Computing, 88 (2010), pp. 111–129.

Chapter 4 is submitted and available as

> [13]: Peter Benner, Thomas Mach: The LR Cholesky algorithm for symmetric hierarchical matrices, Max Planck Institute Magdeburg Preprint MPIMD/12-05, February 2012. 14 pages.

Chapter 5 has be published in

> [12]: Peter Benner, Thomas Mach: Computing all or some eigenvalues of symmetric $\mathcal{H}_\ell$-matrices, SIAM Journal on Scientific Computing, 34 (2012), pp. A485–A496.

and main parts of Chapter 6 have been published as

> [14]: Peter Benner, Thomas Mach: The preconditioned inverse iteration for hierarchical matrices, Numerical Linear Algebra with Applications, 2012. 17 pages.

CHAPTER

ONE

INTRODUCTION

The investigation of eigenvalue problems, see Definition 2.1.6, is one of the core topics of numerical linear algebra. It is guessed that roughly 40% of the papers in SIAM Journal on Matrix Analysis (SIMAX) deal with eigenvalue problems [45].

If we are able to compute the eigenvalues of a matrix, then we can solve a wide range of problems like:

- vibrational analysis,
- ground states in density functional theory,
- stationary distributions of Markov chains,
- and many more.

The eigenvalue problem for unstructured (symmetric) matrices seems to be almost solved. There have, however, been new eigenvalue algorithms for structured matrices in the last decades. These algorithms are divided into two groups. On the one hand, there are algorithms preserving the structure of the spectrum, like the symplectic algorithms in [8, 9]. On the other hand, there are algorithms using the structure of the matrix to accelerate the computations, like the eigenvalue algorithms for semiseparable matrices, e.g., [39, 78, 88]. Here we will present eigenvalue algorithms of the second kind. We will exploit the structure of hierarchical matrices for the construction of new, faster, algorithms.

Hierarchical matrices are data-sparse. They do not only have $\mathcal{O}(n)$ non-zero entries, like sparse matrices, but they can be represented with an almost linear amount of storage. This is seen in the discretization of

$$\lambda f(x) = \int_\Gamma g(x,y) f(y) dy$$

which leads to

$$M_{ij} = \int_\Gamma \int_\Gamma g(x,y)\phi_i(x)\phi_j(y)dxdy.$$

If the kernel-function has a non-local support, then M is dense, even for basis functions ϕ with local support. The exact definitions follow in Section 2.2. Further, if the kernel function $g(x, y)$ can be approximated by a piecewise separable function, then matrix M is an $\mathcal{H}$-matrix. Many kernel functions have such an approximation, which can be computed by interpolation or Taylor expansion. If the kernel function is separable, then M is a separable matrix. As such there is a connection between semiseparable and hierarchical matrices. We will use a theorem on the structure preservation of semiseparable matrices under LR Cholesky transformation [88] to explain the behavior of hierarchical matrices. The hierarchical semiseparable matrices are a subset of hierarchical matrices and a superset of semiseparable matrices. A further interesting subset of the hierarchical matrices are the tridiagonal matrices. We will use an algorithm described as being for tridiagonal matrices [87] for $\mathcal{H}_\ell$-matrices in Chapter 5. In Section 2.5 we explain the relations in detail.

In [45] the eigenvalue problems are classified by their main properties. We already mentioned that we assume the special data-sparse structure of hierarchical matrices. Further, we will restrict the following investigations to real, symmetric matrices. The symmetric eigenvalue problem for real matrices is somehow easier than the general one but the problem is still interesting enough since many applications lead to symmetric matrices, e.g., as mentioned previously. In the preface of [87], B.N. Parlett explains that the differences between symmetric and general eigenvalue problems make it advantageous to treat them separately.

The last point of G. Golub and H. Van der Vorst's classification in [45] is the question; which eigenvalues are required? Here we are not satisfied with computing only the largest or smallest eigenvalue in magnitude, but we also want to have some or all inner eigenvalues.

This special structured eigenvalue problem has been investigated by Hackbusch and Kreß [60], Delvaux, Frederix, and Van Barel [34], and Gördes [47]. Their eigenvalue algorithms will be reviewed in Section 2.6.

In the remainder of this chapter we introduce some notation that have not been mentioned in the list of symbols and explain the structure of this thesis.

1.1 Notation

The notation in the following chapters is mainly following Householder notation. That means: matrices are denoted by capital letters M, N, A, B, lower roman letters x, v, w stand for column vectors, Greek letters λ, μ, ν denote scalar variables or, i.e., $\mu(M)$, scalar valued functions, and B^T stands for the transpose of the matrix B. This is especially the case in indices of matrices where the colon notation is used, where $i : n := \{1, \ldots, n\}$. With M_{ij} we denote the element of M in the i-th row and j-th column. So $M_{i:j,k:l}$ is a submatrix of M.

Further, we follow the usual notation in the field of hierarchical matrices. So we reserve the matrices A and B for low-rank factorizations in the form AB^T. The value k often stands for the rank of AB^T.

Sometimes the symbol $k \ll n$ is used to indicate that k is much lower than n. If $k \ll n$, then $k < n$ and $k \in o(n)$.

The absolute resp. relative errors of computed vectors of eigenvalues are measured using the maximal distance within the set of the eigenvalues:

$$e_{\text{abs}} = \left\| \lambda_i - \hat{\lambda}_i \right\|_\infty \qquad \text{resp.} \qquad e_{\text{rel}} = \left\| \frac{\lambda_i - \hat{\lambda}_i}{\lambda_i} \right\|_\infty .$$

1.2 Structure of this Thesis

In the next chapter known definitions, facts, theorems and properties are collected. We first explain the eigenvalue problem and follow this with a list of the used dense matrix algorithms and their complexity. We review the hierarchical matrix format following the definitions in [48]. The weak admissible condition leads to simple structured $\mathcal{H}$-matrices. Afterwards, we present the examples used in the subsequent chapters. The basics are completed by a short review of the related matrix formats and existing eigenvalue algorithms for hierarchical matrices.

Chapter 3 deals with the QR decomposition of $\mathcal{H}$-matrices. A new QR decomposition is presented. The comparison with two existing QR decompositions, see [6, 73], shows that none of the three are superior in all cases. We try to use the QR decomposition in Chapter 4 for a QR-like algorithm for hierarchical matrices. In Chapter 4 we explain why this leads to an inefficient algorithm for $\mathcal{H}$-matrices but an efficient algorithm for $\mathcal{H}_\ell$-matrices. Therefore, we use a new, more constructive proof for the theorem in [88].

A bisectioning method is used in Chapter 5 to compute the eigenvalues of $\mathcal{H}_\ell$-matrices. This method was original described for tridiagonal matrices in [87]. In each bisection step one has to compute an LDL^T factorization. For $\mathcal{H}_\ell$-matrices this can be done without truncation in linear-polylogarithmic complexity. The algorithm computes one eigenvalue in $\mathcal{O}(k^2 n\,(\log_2 n)^4)$ flops. The algorithm can also be used in $\mathcal{H}$-matrices, but there is no bound on the error or the complexity.

In Chapter 6 we use the preconditioned inverse iteration [68] for the computation of the smallest eigenvalues of $\mathcal{H}$-matrices. Together with the folded spectrum method [104], inner eigenvalues can also be computed. We compare the different algorithms in Chapter 7 and summarize this thesis in a final conclusion chapter.

CHAPTER

TWO

BASICS

Contents

The aim of this chapter is to collect well known basic facts, definitions, lemmas, theorems, and examples necessary for the understanding of the following chapters. Since everything is well known, the lemmas and theorems will mostly be given without proofs. Further, a review of existing algorithms for the computation of eigenvalues of hierarchical matrices is given. At the end of the chapter we briefly describe the compute cluster Otto used for the numerical computations.

2.1 Linear Algebra and Eigenvalues

In the following section we summarize the known facts on matrices, eigenvalues, and standard dense matrix algorithms. This section gives a narrow excerpt of the wide mathematical field of linear algebra. We focus on those aspects important for the thesis.

First we require the definitions of a vector and a matrix, since they are important concepts in linear algebra.

Definition 2.1.1: (vector)
An n-tupel v of scalar variables

$$\begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} =: v$$

is called a *vector*. The space of vectors with n real entries is called $\mathbb{R}^n$, we say $v \in \mathbb{R}^n$.

In the $\mathcal{H}$-matrix context it is advantageous to construct vectors over general index sets $I \subset \mathbb{N}$ and not only over $\{1, \ldots, n\}$. The set of vectors $(v_i)_{i \in I}$ is denoted by $\mathbb{R}^I$, cf. [56].

Definition 2.1.2: (matrix, symmetric matrix)
With $\mathbb{R}^{n \times m}$ we denote the vector space of *real matrices of dimension $n \times m$.*

$$M \in \mathbb{R}^{n \times m} \Rightarrow M = \begin{bmatrix} M_{11} & \cdots & M_{1m} \\ \vdots & \ddots & \vdots \\ M_{n1} & \cdots & M_{nm} \end{bmatrix}$$

Like for vectors, we will use $M \in \mathbb{R}^{I \times J}$ for the matrix

$$M = (M_{ij})_{i \in I, j \in J},$$

with the index sets I for the rows and J for the columns, cf. [56].

If $M_{ij} = M_{ji}$ for all pairs (i, j), then M is called *symmetric.*

If nothing else is stated, then $M \in \mathbb{R}^{n\times n}$ resp. $M \in \mathbb{R}^{I\times I}$, with $|I| = n$ and $M = M^T$. This restriction to symmetric matrices is a strong simplification, but as we have seen in the previous chapter, symmetric problems are of particular interest in applications for mechanics and physics.

Each matrix has a rank. The concept of the rank is important for the hierarchical matrices as we will approximate almost all of their submatrices by low-rank matrices.

Definition 2.1.3: [46]
The *rank* of a matrix $N \in \mathbb{R}^{n\times m}$ is the dimension of the image of N

$$\operatorname{rank}(N) = \dim(\operatorname{im}(N)).$$

Corollary 2.1.4: The rank of $N \in \mathbb{R}^{n\times m}$ is equal to the number of non-zero singular values of N, see also Section 2.1.2,

$$\operatorname{rank}(N) = |\{\sigma_i | 0 < \sigma_i \in \Sigma(N)\}|\,.$$

If N is a square matrix ($m = n$), then the rank is also equal to the number of non-zero eigenvalues

$$\operatorname{rank}(N) = |\{\lambda_i | 0 \neq \lambda_i \in \Lambda(N)\}|\,.$$

Since one can not determine the exact rank in finite precision arithmetic one uses the numerical rank of a matrix instead.

Definition 2.1.5: [15]
The *numerical rank with respect to a threshold* τ of N is the number of singular values larger than σ_1/τ.

The SVD, see Section 2.1.2, can be used to compute the (numerical) rank of N. Besides this, the rank can be computed by special implementations of the QR decomposition, taking care of correct rank computations like, for example, the rank-revealing QR decomposition [25]. The rank revealing QR for $N \in \mathbb{R}^{n\times m}$ has a complexity of $\mathcal{O}(rnm)$, with $r = \operatorname{rank}(N)$.

The next subsection lists the basic facts of the matrix eigenvalue problem.

2.1.1 The Eigenvalue Problem

Definition 2.1.6: (eigenpair) [107]
Let $M \in \mathbb{R}^{n\times n}$ be a matrix. A pair (λ, v) with $v \in \mathbb{C}^n \setminus \{0\}$ and $\lambda \in \mathbb{C}$ that fulfills

$$Mv = \lambda v \tag{2.1}$$

is called an *eigenpair* of M. The vector v is called an *eigenvector* and λ an *eigenvalue*. The set of all eigenvalues of M $\{\lambda_1, \ldots, \lambda_n\}$ is called the *spectrum* and is denoted by $\Lambda(M)$. Sometimes the column vector $\left[\lambda_1, \ldots, \lambda_n\right]^T$ or the diagonal matrix $\operatorname{diag}(\lambda_1, \ldots, \lambda_n)$ are also called $\Lambda(M) \in \mathbb{R}^n$. It is clear from the context whether the set or the column vector is intended.

The computation problem of some or all eigenpairs is called the *eigenvalue problem (EVP)*, e.g., [107, 87, 3].

Remark 2.1.7: The eigenvalue problem is, in general, only solvable by iterative algorithms, since otherwise this would be a contradiction of the Abel–Ruffini theorem [45].

The following lemmas deal with invariance of the spectrum under similarity and congruence transformations. These invariances are the basis for the QR algorithm, see Chapter 4, and for the slicing algorithm, see Chapter 5.

Lemma 2.1.8: [87, Fact 1.1]
Let $M \in \mathbb{R}^{n\times n}$ and $P \in \mathbb{R}^{n\times n}$ be invertible. Then the *similarity transformation* $P^{-1}MP$ preserves the spectrum,

$$\Lambda(M) = \Lambda(P^{-1}MP).$$

If P is further *orthogonal*, $P^TP = I$, then the transformation

$$\Lambda(M) = \Lambda(P^{-1}MP) = \Lambda(P^TMP)$$

is called an *orthogonality transformation.*

Theorem 2.1.9: (Sylvester's inertia law, e.g., [87, p. 11])
Let $M = M^T \in \mathbb{R}^{n\times n}$ and $G \in \mathbb{R}^{n\times n}$ invertible. Then for the *congruence transformation* it holds, that

$$\nu(M) = \nu(G^TMG),$$

with $\nu(M) := |\{\lambda \in \Lambda(M) | \lambda < 0\}|$ the number of negative eigenvalues. The rank of M and the number of positive eigenvalues is also preserved. The triple number of negative, zero, and positive eigenvalues is called *inertia* of M.

Further, the *trace* of the matrix M, which is defined by

$$\operatorname{tr}(M) := \sum_{i=1}^{n} m_{ii} = \sum_{i=1}^{n} \lambda_i, \tag{2.2}$$

will be required.

Here we will only investigate the symmetric eigenvalue problem, where M is a symmetric matrix $M = M^T$. In case where the matrix is symmetric, the eigenvalue problem is easier since the following lemmas hold.

Lemma 2.1.10: [87, Fact 1.2]
Let $M \in \mathbb{R}^{n\times n}$ be a symmetric matrix and (λ, v) an eigenpair of M. Then it holds, that $v \in \mathbb{R}^n$ and $\lambda \in \mathbb{R}$.

So we can handle the symmetric eigenvalue problem completely in the field of reals.

Lemma 2.1.11: (spectral theorem) [87, Fact 1.4]
For any symmetric matrix $M \in \mathbb{R}^{n\times n}$, there is an orthogonal matrix V, so that

$$M = V\Lambda(M)V^T = \sum_{i=1}^{n} \lambda_i v_i v_i^T.$$

The columns v_i of V form an ortho-normalized set of the eigenvectors of M.

The *condition number* $\kappa(V)$ is defined by

$$\kappa(V) = \|V\|_2 \left\|V^{-1}\right\|_2. \tag{2.3}$$

A large condition number $\kappa(V)$ means that the system of equations $Vx = b$ is *ill-conditioned*. The solution x is then very sensitive to perturbations in b. The following lemma states that the symmetric eigenvalue problem is well conditioned.

Lemma 2.1.12: [46, Corollary 7.1-3]
If $M \in \mathbb{R}^{n\times n}$ is normal ($M^TM = MM^T$), then the condition number of the eigenvalue problem of M is

$$\kappa_{\text{EVP}}(M) := \kappa(V) = 1,$$

where V is the matrix of eigenvectors from Lemma 2.1.11.

In the symmetric case all the eigenvalues are real. As such, we will use the following ordering:

$$\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n.$$

2.1.2 Dense Matrix Algorithms

In this subsection the main properties of algorithms for dense matrices are discussed. The dense algorithms play two roles here. On the one hand, a special algorithm for a data-sparse structure should have an advantage over the corresponding dense algorithm, since otherwise there is no need for the special algorithm. This is particularly reflected through comparisons between the dense QR algorithm and the eigenvalue algorithms for hierarchical matrices. On the other hand, dense algorithms are used on the lowest level of the hierarchical structure in the hierarchical arithmetic, see next section.

In this thesis the terms dense, sparse and data-sparse matrix are used frequently. The following definition distinguishes these types of matrices.

Definition 2.1.13: (sparse, dense and data-sparse matrix)
A matrix $M \in \mathbb{R}^{n\times n}$ is called *sparse*, if there is a constant $c \ll n$, so that

$$|\{i|M_{ij} \neq 0\}| \leq c \quad \forall j = 1, \dots, n.$$

One needs at most $\mathcal{O}(cn)$ entries of storage for M if one stores only the non-zero entries.

If the number of non-zero entries in M is large ($c \in \mathcal{O}(n)$), then M is called *dense* and should be stored in a dense matrix format requiring n^2 storage-entries.

If there is a representation of (the dense matrix) M, which requires only $\mathcal{O}(n\,(\log_2 n)^\beta)$ storage entries, where β is a positive real constant independent of n, then we will call this representation of M a *data-sparse representation*.

Further, if a vector $v \in \mathbb{R}^n$ consists of only $c \ll n$ non-zero entries, then the vector is called *sparse*, too.

The set of pairs (i, j), with $M_{i,j} \neq 0$ is called the *sparsity pattern* of M. Sometimes, for $M \in \mathbb{R}^{I\times J}$, this set is regarded as the edge set of a graph with the node set $\{(i, j)|i \in I, j \in J\}$, [92]. Analog for a vector v, we call

$$\{i \in I|v_i \neq 0\},$$

the sparsity pattern of $v \in \mathbb{R}^I$.

Definition 2.1.14: (bandwidth, tridiagonal matrix, diagonal matrix) [92]
Let $b \in \{0, \dots, n-1\}$. A matrix $M \in \mathbb{R}^{n\times n}$ is called *band matrix* if there is no non-zero entry M_{ij} with $|i-j| > b$. The matrix M has then the bandwidth b. A matrix of bandwidth 1 is called *tridiagonal matrix*. *Diagonal matrices* have the bandwidth 0.

If $b \ll n$, then a band matrix with bandwidth b is also sparse.

Definition 2.1.15: (Hessenberg form) [107]
A matrix $N \in \mathbb{R}^{n\times n}$ is called to be of *upper Hessenberg form* if $N_{ij} = 0$ for all i, j with $i > j + 1$. The matrix N is of *lower Hessenberg form* if N^T is of upper Hessenberg form.

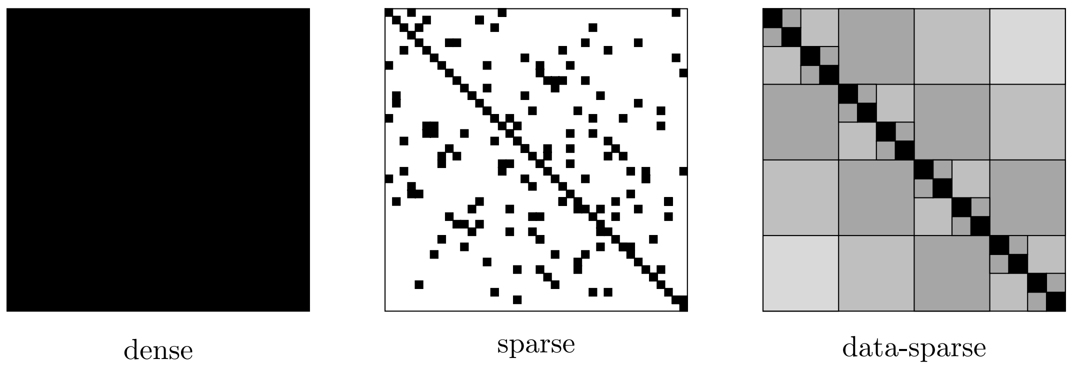


Figure 2.1: Dense, sparse and data-sparse matrices.

A symmetric matrix in upper or lower Hessenberg form is tridiagonal.

In the remainder of this subsection we discuss the dense matrices. We focus on the complexity of the storage and the algorithms. In the later comparison with the $\mathcal{H}$-arithmetic we will see that for matrices large enough, the $\mathcal{H}$-arithmetic is more efficient than the dense matrix arithmetic.

Storage If M is sparse, then M can be stored in a sparse matrix format in $\mathcal{O}(n)$. For instance one can store for each non-zero entry of M_{ij} a triple (i, j, M_{ij}) as it is done in the sparse matrix format used by MATLAB to display sparse matrices.[1] If M is dense, then the costs of storing M in a sparse matrix format will exceed the costs of the dense matrix format.

In the dense matrix format all entries of a matrix are stored column by column, row by row, or sometimes in a special blocking. In each case one has to store n^2 elements for $M \in \mathbb{R}^{n \times n}$, so the storage complexity is $\mathcal{O}(n^2)$. Storing column by column is used in the Fortran-based implementation of BLAS.

In the next but one section, we will see that $\mathcal{H}$-matrices are dense but data-sparse, so that neither the sparse nor the dense matrix formats are efficient for them.

Matrix-Vector Product To compute $y = Mx$ with $x, y \in \mathbb{R}^n$ and $M \in \mathbb{R}^{n \times n}$, one requires $2n^2 - n$ flops.

Matrix-Matrix Product The computation of AB, with $A \in \mathbb{R}^{n \times k}$ and $B \in \mathbb{R}^{k \times m}$, requires $2nmk$ flops. If $m, k \in \mathcal{O}(n)$, then $\mathcal{O}(n^3)$ flops are necessary for naive implementations. Sophisticated implementations can reduce the effort to $\mathcal{O}(n^{\geq 2.376})$ flops, see [31].

[1] There are more efficient sparse matrix formats, like compressed sparse column, which MATLAB uses for the sparse computations [43].

The dense matrix-matrix multiplications used in the numerical examples are performed by BLAS and cost $\mathcal{O}(n^3)$ flops.

QR Decomposition The QR decomposition is an important tool in the field of matrix computations. Every matrix $N \in \mathbb{R}^{n\times m}$ can be factorized in an orthogonal matrix $Q \in \mathbb{R}^{n\times n}$ and an upper triangular $R \in \mathbb{R}^{n\times m}$ (or trapezoidal if $m < n$),

$$N = QR = [\square\,◹]\,. \tag{2.4}$$

This factorization is called the *QR decomposition* of N. The computation of the decomposition consists of several Householder rotations and requires $3n^2(m - n/3)$ flops [46]. Further properties of the QR decomposition can be found in many textbooks on numerical linear algebra, e.g., in [37, 46].

LU Decomposition Let $N \in \mathbb{R}^{n\times n}$. If all leading principal submatrices of N are invertible, then N has also an LU decomposition in a lower triangle with unit diagonal L and upper triangle U,

$$N = LU = [◺\,◹]\,.$$

The LU decomposition is related to Gaussian elimination and requires $\mathcal{O}(n^3)$ flops. One should use pivoting for stability reasons, see, e.g., [64].

Cholesky Decomposition If $M = M^T > 0$, then one can find a symmetric factorization similar to the LU decomposition with

$$M = LL^T = [◺\,◹]\,,$$

where L is a lower triangular with a diagonal of positive entries. Due to the symmetry, the computation of the Cholesky factoriazation is cheaper than the LU decoomposition but still of cubic complexity. Here we are permitted to omit pivoting, since M is positive definite [64].

LDL^T Decomposition

Definition 2.1.16: (LDL^T factorization [46])
If $M \in \mathbb{R}^{n\times n}$ is a symmetric, but not necessarily positive definite, matrix and all the leading principal submatrices of M are invertible, then there exists a unit lower triangular matrix L and a diagonal matrix $D = \operatorname{diag}(d_1, \ldots, d_n)$ such that

$$M = LDL^T = [◺ \diagdown ◹]\,.$$

This factorization is called *LDL^T factorization* or *LDL^T decomposition.*

The computation of LDL^T factorizations is of cubic complexity. Theorem 2.1.9 tells us that M and D have the same inertia.

QR Algorithm The implicit, multi-shift QR algorithm, based on the invention by Francis [41, 42], is the most widely used algorithm for the computation of all eigenvalues of small dense matrices. Modern implementation used now in LAPACK/MATLAB are based on aggressive early deflation by R. Byers et al., see [22]. The QR algorithm is of cubic complexity [3], since the transformation to upper Hessenberg resp. tridiagonal form needs $\mathcal{O}(n^3)$ flops. The explicit QR algorithm is explained in Subsection 4.1.2.

Eigenvalue Algorithms for Symmetric Matrices The symmetric matrix M is first transformed into tridiagonal form, which costs $\mathcal{O}(n^3)$ flops. The eigenvalues of tridiagonal matrices can be computed by the QR algorithm, too. Further, one can use the divide-and-conquer algorithm [32], the modern implementation of Rutishauser's qd algorithm [89], the dsqd algorithm [40], or the MR^3 algorithm [108]. These algorithm are of quadratic complexity.

An alternative is the transformation of M into a semiseparable matrix, see, e.g, [103], which also costs $\mathcal{O}(n^3)$ flops. For semiseparable matrices there are efficient algorithms based on the implicit QR algorithm [102], the divide-and-conquer algorithm [78], and the LR Cholesky algorithm [88]. The complexity is dominated by the transformation into semiseparable form.

Singular Value Decomposition Each matrix $N \in \mathbb{R}^{n\times m}$ has a *singular value decomposition* (SVD)

$$N = U\Sigma V^T = [\square \diagdown \square], \tag{2.5}$$

with orthogonal matrices $U \in \mathbb{R}^{n\times k}$, $V \in \mathbb{R}^{m\times k}$ and a diagonal matrix $\Sigma \in \mathbb{R}^{k\times k}$, see [44]. The entries σ_i of D are positive and are called *singular values*. The σ_i are in decreasing order. The number of singular values is $k = \min\{n, m\}$. If some singular values are zero, then one can also compute the thin SVD [46], where only the singular vectors for nonzero singular values are stored in U and V:

$$N = \begin{bmatrix} U_1, U_2 \end{bmatrix} \begin{bmatrix} \Sigma_1 & \\ & 0 \end{bmatrix} \begin{bmatrix} V_1^T \\ V_2^T \end{bmatrix} = U_1 \Sigma_1 V_1^T.$$

By setting the smallest singular values to zero, one can compute the best approximation in the 2- and Frobenius-norms.

Theorem 2.1.17: [46, p. 72], [64]
Let $N = U\Sigma V^T \in \mathbb{R}^{n\times n}$ be the singular value decomposition of N, with $r = \operatorname{rank}(N)$. If $k < r$ and $N_k = \sum_i^k \sigma_i u_i v_i^T$, then

$$\min_{S\in\mathbb{R}^{n\times n},\ \operatorname{rank}(S)=k} \|N - S\|_2 = \|N - N_k\|_2 = \sigma_{k+1},$$

$$\min_{S\in\mathbb{R}^{n\times n},\ \operatorname{rank}(S)=k} \|N - S\|_F = \|N - N_k\|_F = \left(\sum_{i=k+1}^{r} \sigma_i^2\right)^{\frac{1}{2}}.$$

The computation of the SVD costs, in general, $\mathcal{O}(n^3)$ flops [46].

Corollary 2.1.18: Let $N \in \mathbb{R}^{n\times n}$ with $\operatorname{rank}(N) = k$. Then there are matrices $A, B \in \mathbb{R}^{n\times k}$, with

$$N = AB^T =$$.

Proof. There is a singular value decomposition of N with k nonzero singular values.

$$N = \sum_{i}^{k} \sigma_i u_i v_i^T$$

Set

$$A = \begin{bmatrix} u_1 & u_2 & \cdots & u_k \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} \sigma_1 v_1 & \sigma_2 v_2 & \cdots & \sigma_k v_k \end{bmatrix}.$$

■

All these algorithms are well known. However, except the matrix-vector product, all these algorithms are of cubic complexity in time and require a quadratic amount of storage. This means that a matrix of size $16\,384 \times 16\,384$ requires 2 GB of storage and a computation of the eigenvalues by the QR algorithm will take about one hour on a standard desktop machine. Thus, on today's single core Intel® Pentium® D 3.00 GHz the computation will take 53 minutes. A matrix of size $32\,768 \times 32\,768$ would take 8 GB RAM and about 8 hours to compute all eigenvalues and this is already too much for many of today's desktop machines.

It is not advisable to handle larger matrices in the dense matrix format. So it is worth looking deeper into the hierarchical matrix format, since this enables us to handle larger dense matrices approximately. Before this, we should have a look at integral operators, since they are a field of application for $\mathcal{H}$-matrices.

2.2 Integral Operators and Integral Equations

The discretization of non-local integral operators leads to dense matrices, which often have good $\mathcal{H}$-matrix approximations. E.g., integral operators occur in the inversion of (partial) differential operators or in population dynamics [72]. The discretization of integral equations leads to linear systems of equations [54].

2.2.1 Definitions

For simplification we restrict ourselves to the linear Fredholm integral equations of first kind, see [54] for the classification of integral equations.

Definition 2.2.1: (integral operator, kernel function)
Let $\Omega \subset \mathbb{R}^d$. The operator $F(\cdot)$, defined by

$$F(\phi) := \int_\Omega g(x,y)\phi(y)dy,$$

is called an *integral operator*. The funktion $g(\cdot,\cdot)$ is called the *kernel function* of $F(\cdot)$. The equation

$$f(x) = \int_\Omega g(x,y)\phi(y)dy$$

is called *linear Fredholm integral equations of first kind.*

The discretization of F using the basis sets $(\phi_i)_{i\in I}$ and $(\psi_j)_{j\in J}$ leads to the matrix

$$M_{ij} := \int_\Omega \psi_j(x)F(\phi_i(x))dx \qquad i \in I, j \in J.$$

Often, basis functions have a local support like linear hat-functions. If the kernel function $g(x,y)$ is non-zero only for $x \in B_c(y)$ and the basis functions are sufficiently uniformly distributed over Ω, then the matrix M is sparse. Otherwise, if $g(x,y)$ is a non-local kernel ($g(x,y) \neq 0$ for almost all (x,y)) then M is dense.

Definition 2.2.2: (separable function) [54, Definition 3.3.3]
A (kernel) function $\tilde{g}(x,y)$ is called *separable*, if there are functions $\zeta_\nu(x)$ and $\xi_\nu(y)$, so that

$$\tilde{g}(x,y) = \sum_{\nu=1}^{k} \zeta_\nu(x)\xi_\nu(y).$$

In the context of approximating the kernel $g(x,y)$ by a separable function $\tilde{g}(x,y)$, the kernel $\tilde{g}(x,y)$ is called a degenerate approximation of $g(x,y)$. However, there are separable kernels whose discretization leads to separable matrices.

Lemma 2.2.3: [56, p. 75]
Let M be the discretization of an integral operator $F(\cdot)$, as in Definition 2.2.1. If the kernel function $g(x,y)$ is separable,

$$g(x,y) = \sum_{\nu=1}^{k} \zeta_\nu(x)\xi_\nu(y),$$

then M is a matrix of rank k and can be written in the form

$$M = AB^T.$$

Sometimes the kernel function g is inseparable but can be approximated by a separable function $\tilde{g}$ on the subdomain $\Omega_t \times \Omega_s$, with $\Omega_t, \Omega_s \subset \Omega$. Let t be the set of basis functions ψ_i with $\operatorname{supp}(\psi_i) \subset \Omega_t$ and s the set of basis functions ϕ_j with $\operatorname{supp}(\phi_j) \subset \Omega_s$. Then the submatrix $M|_{t\times s}$ is of low rank.

The goal is to approximate almost all submatrices of M by such low rank approximations. In Section 2.3 we will see how the hierarchical matrix format can be used to find and handle such submatrices efficiently. But before this, we will have a look at the boundary element method and the integral operators which occur there.

2.2.2 Example - BEM

The boundary element method (BEM) is a method for solving partial differential equations. Compared with the finite element method (FEM), one does only require basis functions for the discretization of the boundary of the domain. This reduces the number of basis function since the boundary of a d-dimensional domain Ω is only $(d-1)$-dimensional.

Lemma 2.2.4: Let $\Omega \in \mathbb{R}^3$ be a connected space with a connected boundary $\delta\Omega$. The function

$$\phi(x) = \frac{1}{4\pi} \int_{\delta\Omega} \frac{f(x)}{\|x-y\|_2} dy, \tag{2.6}$$

with $f : \delta\Omega \to \mathbb{R}$, fulfills the equation

$$\Delta\phi(x) = 0 \quad \forall x \in \Omega.$$

Proof. [54, Lemma 8.1.3]. ∎

Remark 2.2.5: There are generalizations of this lemma for other differential operators, other dimensions $d \neq 3$, and non connected boundaries, see, e.g., [93].

This lemma transforms a differential equation on Ω into the integral equation (2.6) on $\delta\Omega$. The discretization of Equation (2.6) yields

$$Mx = b, \quad M_{ij} = \frac{1}{4\pi} \int_{\delta\Omega} \int_{\delta\Omega} \frac{\phi_i(x)\phi_j(y)}{\|x-y\|_2} dy dx, \tag{2.7}$$
$$b_i = \int_{\delta\Omega} f(x)\phi_i(x) dx.$$

The kernel function

$$\frac{1}{4\pi\,\|x-y\|_2}$$

is smooth enough to be approximated by a piecewise separable function, except in the case where $x = y$. For instance, these approximations can be done by Taylor expansions, see [75] for example. So we can approximate the discretization matrix M of this problem through a hierarchical matrix. In the next section the set of hierarchical matrices is introduced. After that section, in Subsection 2.4.2 we will revisit this example again.

2.3 Introduction to Hierarchical Arithmetic

2.3.1 Main Idea

Hierarchical matrices were introduced by Hackbusch in 1998 [55]. The concept of hierachical, or $\mathcal{H}$-, matrices for short, is based on the observation that submatrices of a full rank matrix may be of low rank resp. have low rank approximations.

This observation was already used in, e.g., the fast multipole method (FMM) [52], in the panel clustering [61], or in the matrix-skeleton approximation [98].

There is a second important observation: The inverses of finite element matrices have, under certain assumptions, submatrices with exponentially decaying singular values. This means that these submatrices have also good low rank approximations.

Example 2.3.1: Let

$$\Delta_h = \begin{bmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & 2 & -1 & \ddots & \vdots \\ 0 & -1 & 2 & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & -1 \\ 0 & \cdots & 0 & -1 & 2 \end{bmatrix} \in \mathbb{R}^{64\times 64}$$

and $\Delta_{2,h} = I \otimes \Delta_h + \Delta_h \otimes I$. The matrix $\Delta_{2,h}$ is sparse. Then $\left(\Delta_{2,h}^{-1}\right)_{3585:4096,1:512}$ has only 12 singular values larger than 10^{-12}, see Figure 2.2. The inverse is computed by the MATLAB function `inv` and the singular values by `svd`, once with single precision arithmetic and once with double precision. The singular values smaller than 10^{-15} seem to be perturbed, so we should only trust the singular values shown in Figure 2.3. These singular values show the expected exponential decay.

The hierarchical matrices are superior to these predecessors, since they permit much more algebraic operations like approximate inversion, approximate Cholesky decomposition, and approximate multiplication within the format.

Let $M_{r\times s}$ be a submatrix that admits low rank approximations, then

$$M_{r\times s} \approx AB^T, \quad A \in \mathbb{R}^{r\times k}, B \in \mathbb{R}^{s\times k},$$

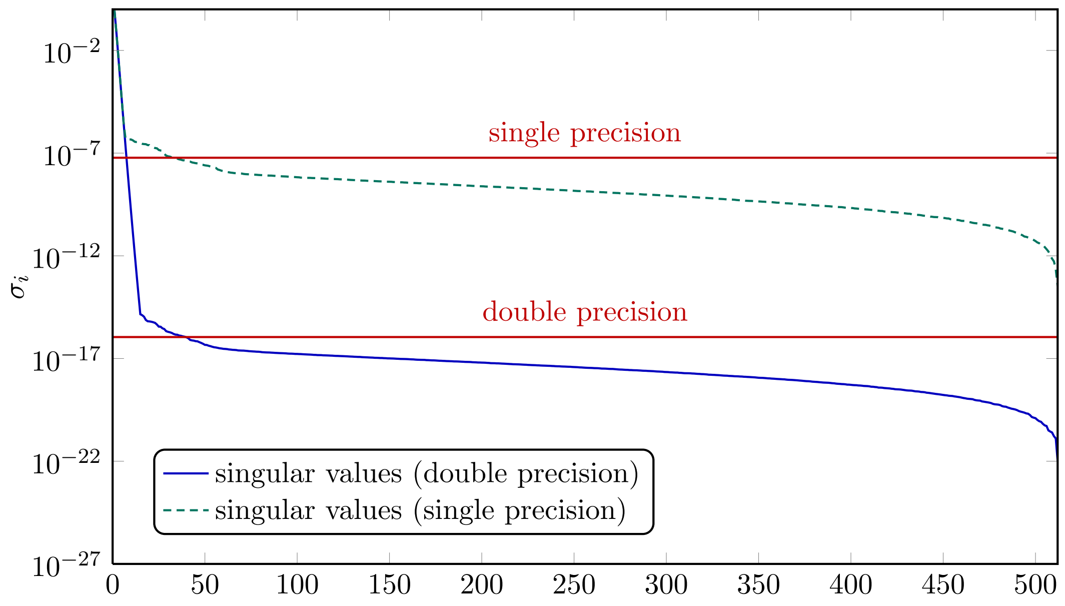


Figure 2.2: Singular values of $\left(\Delta_{2,h}^{-1}\right)_{3585:4096,1:512}$.

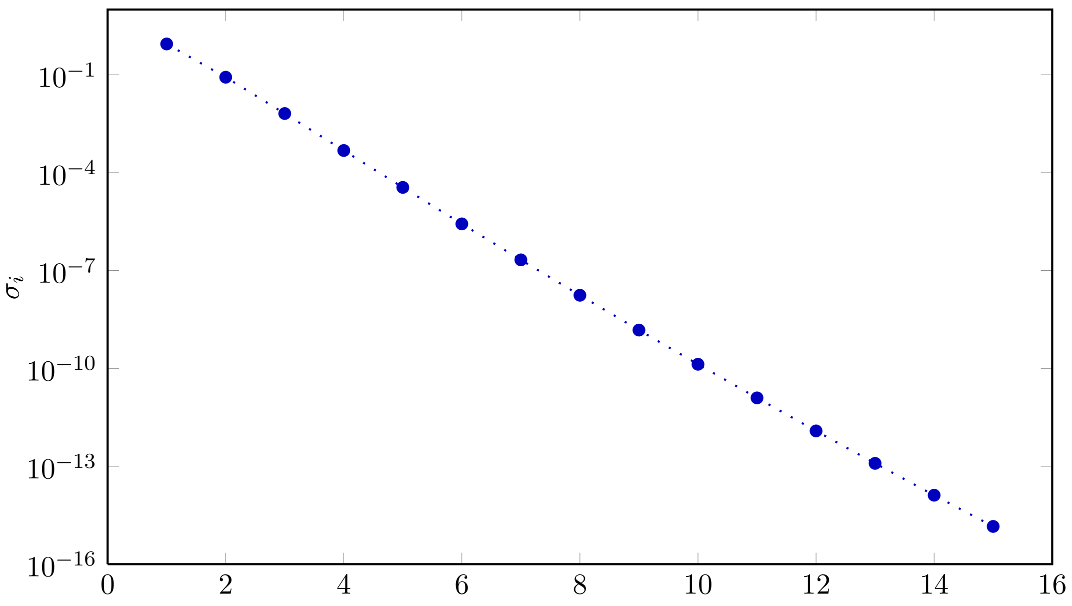


Figure 2.3: Singular values of $\left(\Delta_{2,h}^{-1}\right)_{3585:4096,1:512}$.

with rank k. If k is much smaller than r and s, then storing A, B is much cheaper than storing $M_{r\times s}$.

The definitions in the next subsection introduce all the essential objects necessary for the definition of hierarchical matrices, the hierarchical arithmetic, the understanding of the properties of hierarchical matrices, and for the subsequent chapters.

2.3.2 Definitions

We follow the notation of [48] and [56], where further explanations can be found. Before we define hierarchical matrices, we need some other definitions first:

Definition 2.3.2: (hierarchical tree, see [48, Definition 3.3])
Let I be an index set. We call a tree $T_I = (V, E)$ with vertices $V \subset \mathcal{P}(I) \setminus \{\emptyset\}$ and edges $E \subset V \times V$ a *hierarchical tree*, or $\mathcal{H}$-tree for short, of the index set I if the following conditions are fulfilled:

- the root of T_I is I,
- a vertex $v \in V$ is either a leave T_I or the disjoint union of its sons:

$$v = \dot{\bigcup_{s\in S(v)}} s.$$

The set of sons of a vertex $r \in T_I$ is denoted by $S(r)$. We define the set of *descendants $S^*(r)$ of a vertex* $r \in T_I$ using [18, Definition 3.5]

$$S^*(r) = \begin{cases} \{r\} & \text{if } S(r) = \emptyset, \\ \{r\} \cup \bigcup_{s\in S(r)} S^*(s) & \text{otherwise.} \end{cases} \tag{2.8}$$

We count the levels beginning with the root of the tree. On level 0 there is only one vertex $I \in T_I^{(0)}$. The set $T^{(i)}$ is defined as the set of the sons of vertices out of $T^{(i-1)}$,

$$T^{(i)} := \left\{ s \in S(r) \middle| r \in T^{(i-1)} \right\} \quad \text{and} \quad T^{(0)} := \{I\}. \tag{2.9}$$

The *depth of the tree* T_I is defined by

$$\text{depth}\,(T) := \max\left\{ i \in \mathbb{N} \middle| T^{(i)} \neq \emptyset \right\}. \tag{2.10}$$

If a cardinality balanced clustering strategy is used, then the depth (T_I) is in $\mathcal{O}(\log_2 n)$ [49, p. 320ff]. The vertices without sons form the set of *leaves of the tree* T_I and are denoted by

$$\mathcal{L}(T_I) := \{v \in T_I | S(v) = \emptyset\}. \tag{2.11}$$

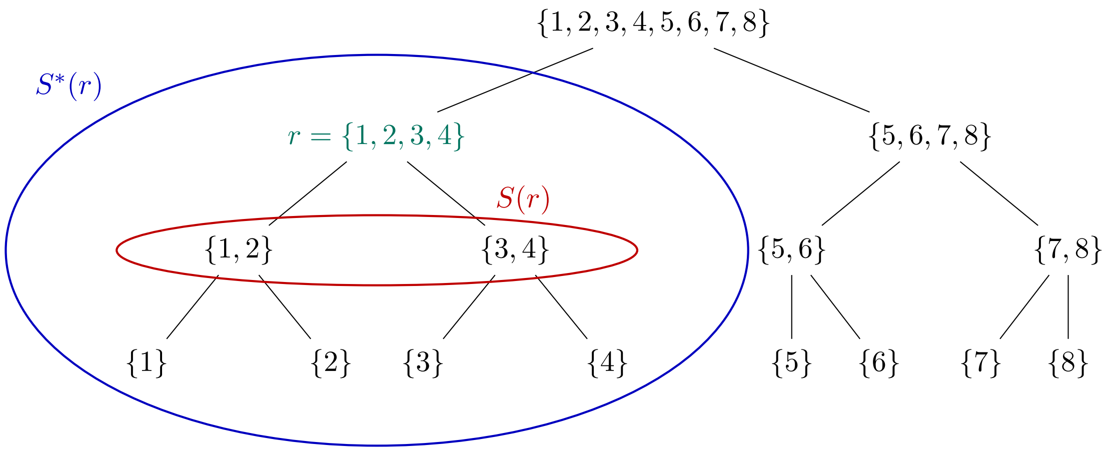
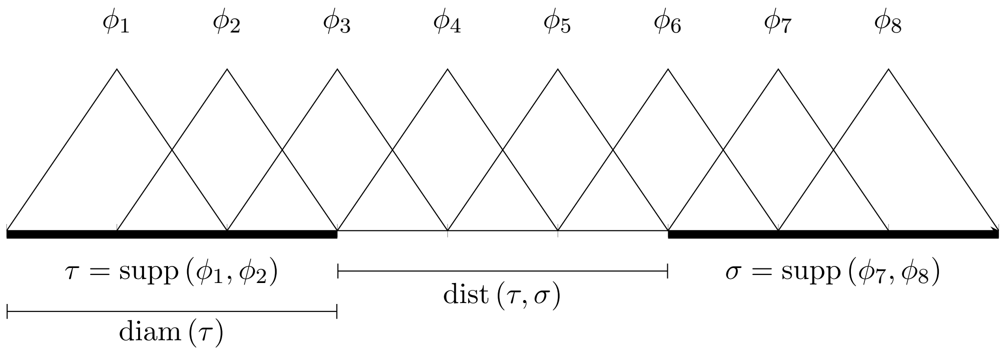


Figure 2.4: $\mathcal{H}$-tree T_I.



ϕ_1
ϕ_2
ϕ_3
ϕ_4
ϕ_5
ϕ_6
ϕ_7
ϕ_8
$\tau = \operatorname{supp}(\phi_1, \phi_2)$
$\operatorname{dist}(\tau, \sigma)$
$\sigma = \operatorname{supp}(\phi_7, \phi_8)$
$\operatorname{diam}(\tau)$

Figure 2.5: Example of basis functions.

The leaves on level i are denoted by

$$\mathcal{L}^{(i)}(T_I) := \mathcal{L}(T_I) \cap T_I^{(i)}. \tag{2.12}$$

A simple example of an $\mathcal{H}$-tree, based on [4, Example 2.3.3], is shown in Figure 2.4.

Remark 2.3.3: (Construction of T_I)
There are different possibilities of constructing the $\mathcal{H}$-tree T_I. We require the basis functions $\phi_i, i \in I$ and their support

$$\operatorname{supp}(\phi_i(x)) := \overline{\{x | \phi_i(x) \neq 0\}}.$$

We have to split the indices in the index set I into two subsets s and t. The indices with neighboring basis functions are grouped together, so that

$$\tau = \Omega_t := \bigcup_{i \in t} \operatorname{supp}(\phi_i) \quad \text{and} \quad \sigma = \Omega_s := \bigcup_{i \in s} \operatorname{supp}(\phi_i)$$

have a small diameter, see Figure 2.5. In the geometrically balanced clustering, one tries to make the diameters of τ and σ equal. Therefore, one divides the domain Ω_I according to the space direction with maximal diameter. Choosing the space direction with maximal diameter is useful in order to prevent Ω_t from degenerating. The splitting is stopped if the minimal block size $n_{\min}$ is reached. For quasi-uniform meshes, $h_{\min} \sim h_{\max}$, it was shown in [49] that

$$\operatorname{depth}(T_I) = \mathcal{O}(\log_2 n).$$

Besides this, there is a cardinality balanced clustering where one tries to make the cardinality of t and s equal. The depth of a cardinality balanced tree is

$$\operatorname{depth}(T_I) \approx \log_2 n - \log_2 n_{\min}.$$

A matrix is built over the product index set $I \times J$. Hence we require a hierarchical partitioning in the form of the block hierarchical tree or block cluster tree. Since the product of two $\mathcal{H}$-trees leads to a dense block hierarchical tree, we require before the admissible condition, which tells us whether a block of M can be approximated by a low rank factorization or if the block needs further subdivision.

Definition 2.3.4: (admissibility condition, [18])
Let $T_{I\times J}$ be a block $\mathcal{H}$-tree. The function

$$\mathcal{A} : T_I \times T_J \to \{\text{true}, \text{false}\}$$

is called *admissibility condition* if

$$\mathcal{A}(r \times s) \Rightarrow \mathcal{A}(r' \times s) \wedge \mathcal{A}(r \times s') \quad \forall r \in T_I, s \in T_J, r' \in S(r), s' \in S(s).$$

A block $b = (r, s)$ with $\mathcal{A}(b) = \text{true}$ is called *admissible*, otherwise *inadmissible*.

The admissibility condition decides which blocks are stored in low rank factorized form.

There are two commonly used admissibility conditions. The standard η-admissibility condition is defined in [48, p. 9]

$$\mathcal{A}_\eta(r, s) = \begin{cases} \text{true}, & \text{if } \min\{\operatorname{diam}(\tau), \operatorname{diam}(\sigma)\} \leq 2\eta \operatorname{dist}(\tau, \sigma), \\ \text{false}, & \text{else}, \end{cases}$$

where $\tau = \bigcup_{i\in r} \operatorname{supp}(\phi_i)$ and $\sigma = \bigcup_{i\in s} \operatorname{supp}(\phi_i)$, with ϕ_i the basis function related to index i. This admissibility condition is sometimes simplified by replacing τ and σ with the minimal axis-parallel bounding boxes.

Further, we will use the weak admissibility condition [58]:

$$\mathcal{A}_{\text{weak}}(r, s) := \begin{cases} \text{true}, & r \cap s = \emptyset, \\ \text{false}, & \text{otherwise}. \end{cases} \tag{2.13}$$

We will now use the admissibility condition for the definition of the hierarchical block tree.

Definition 2.3.5: (hierarchical block tree)
Let T_I and T_J be hierarchical trees. The hierarchical tree $T_{I\times J}$ is called *hierarchical block tree*, short $\mathcal{H}_\times$-tree or block $\mathcal{H}$-tree, or *block cluster tree*, if $\forall b = (r, s) \in T_{I\times J}$:

(i) $r \in T_I \wedge s \in T_J$ and

(ii) If $\mathcal{A}(r, s) = \text{false}$, then $S_{T_{I\times J}}(b) = \{(r', s') \in T_{I\times J} | r' \in S_{T_I}(r) \wedge s' \in S_{T_J}(s')\}$.

(iii) If $\mathcal{A}(r, s) = \text{true}$, then $S_{T_{I\times J}}(b) = \emptyset$.

The set of leaves of $T_{I\times J}$ that are additionally admissible, is denoted by

$$\mathcal{L}^+(T_{I\times J}) := \{r \times s \in \mathcal{L}(T_{I\times J}) | \mathcal{A}(r, s) = \text{true}\} . \tag{2.14}$$

The inadmissible leaves form the set

$$\mathcal{L}^-(T_{I\times J}) := \{r \times s \in \mathcal{L}(T_{I\times J}) | \mathcal{A}(r, s) = \text{false}\} . \tag{2.15}$$

Now we are able to define the set of $\mathcal{H}$-matrices:

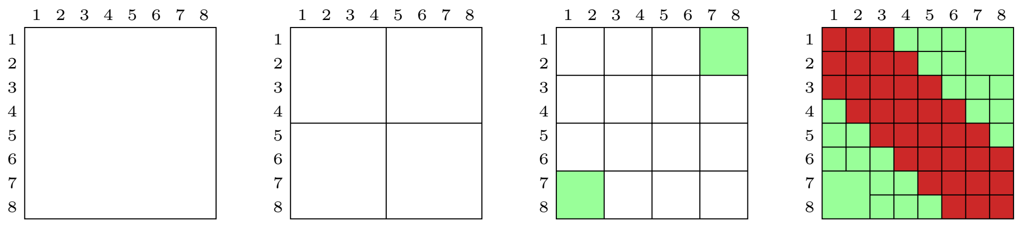
Definition 2.3.6: (hierarchical matrix) [49]
We define the set of *hierarchical matrices* for the block $\mathcal{H}$-tree $T_{I\times J}$ with *admissibility condition* $\mathcal{A}$, maximal *block-wise rank* k and *minimal block size* $n_{\min}$ by

$$\mathcal{H}(T_{I\times J}, \mathcal{A}, k) := \left\{ M \in \mathbb{R}^{I\times J} \left| \begin{array}{l} \forall r \times s \in \mathcal{L}(T_{I\times J}) : \\ \text{if } (\mathcal{A}(r, s) = \text{true}) \text{ then } \operatorname{rank}(M_{r\times s}) \le k, \\ \text{else } \min\{|r|, |s|\} \le n_{\min} \end{array} \right. \right\} .$$

If the standard admissibility condition is used, then we write also $\mathcal{H}(T_{I\times J}, k)$. If the tree is determined by the context, then we also omit the $T_{I\times J}$ and write simply $\mathcal{H}(k)$ for the set of $\mathcal{H}$-matrices.

The admissible blocks of M are, at most, of rank k. We store them in a factorized form $M|_b = AB^T$, with $b \in \mathcal{L}^+(T_{I\times J})$, A, B rectangular matrices with k columns. Matrices of rank k, which are given in the factored form AB^T, are called **R**k-*matrices*. The inadmissible leaves of a block size less than or equal to $n_{\min}$ in one dimension are stored in the dense matrix format, since they neither have a cheap factorization, nor is a further subdivision useful regarding the storage or computational complexity.

A simple example of an $\mathcal{H}$-matrix with the different levels is shown in Figure 2.6a (standard admissibility condition) and in Figure 2.6b (weak admissibility condition). The green blocks are admissible, the red ones are inadmissible. The structure of the $\mathcal{H}$-matrix


(a) Standard admissibility condition $\mathcal{A}_\eta$, $\eta = \frac{1}{4}$.

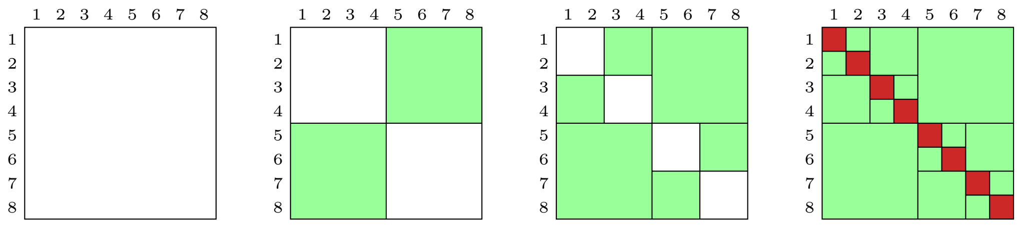

(b) Weak admissibility condition $\mathcal{A}_{\text{weak}}$.

Figure 2.6: $\mathcal{H}$-matrix based on T_I from Figure 2.4.

with the weak admissibility condition is simpler. The $\mathcal{H}$-matrices with weak admissibility condition have special properties. In Subsection 2.3.4 we will investigate that subset of $\mathcal{H}$-matrices.

If nothing else is stated, we will assume that the $\mathcal{H}$-matrices are built over the product index set $I \times I$ by using the same $\mathcal{H}$-tree T_I for the row and column index sets. This is a natural assumption for the eigenvalue problem, since the $\mathcal{H}$-matrix M is a mapping from $\mathbb{R}^I$ to $\mathbb{R}^I$. Otherwise we can not define eigenvalues and eigenvectors of M. Further, we assume M to be symmetric. Using twice the same $\mathcal{H}$-tree T_I ensures that the structure of M is also symmetric. Without this assumption the $\mathcal{H}$-Cholesky and $\mathcal{H}$-LDL^T decomposition of M become much more complicated. Further we assume for simplicity that only binary trees are used with depth $(T) \in \mathcal{O}(\log_2 n)$. This assumption only simplifies the presentation of the arithmetic operations, which can also be formulated for non-binary trees.

The following constants are necessary for the complexity estimation of the $\mathcal{H}$-arithmetic. The constants are properties of the structure.

Definition 2.3.7: (sparsity constant) [49, Definition 2.1]
The *sparsity constant* of an $\mathcal{H}$-matrix is defined as

$$C_{sp} := \max \left\{ \max_{r \in T_I} \left| \{ s \in T_I | \, r \times s \in T_{I \times I} \} \right| , \max_{s \in T_I} \left| \{ r \in T_I | \, r \times s \in T_{I \times I} \} \right| \right\}. \quad (2.16)$$

In [49] it is shown how $T_{I \times I}$ has to be constructed to get $\mathcal{H}$-matrices with sparsity constants C_{sp} independent of the dimension n.

Definition 2.3.8: (idempotency constant) [49, Definition 2.22]
The *idempotency constant* of an $\mathcal{H}$-matrix that is built over a hierarchical product tree $T_{I\times I} = T_I \times T_I$ is defined as

$$C_{id}(\tau\times\sigma) := \left| \left\{ \tau'\times\sigma' \;\middle|\; \begin{array}{l} \tau' \in S^*(\tau), \sigma' \in S^*(\sigma) \text{ and} \\ \exists \rho \in T_I : \tau'\times\rho \in T_{I\times I} \wedge \rho\times\sigma' \in T_{I\times I} \end{array} \right\} \right|$$

$$C_{id} := \max_{\tau\times\sigma\in T_{I\times I}} C_{id}(\tau\times\sigma) \tag{2.17}$$

Lemma 2.3.9: [49, Lemma 2.4]
Let $M \in \mathcal{H}(T_{I\times I}, k)$ be an $\mathcal{H}$-matrix with sparsity constant C_{sp}, with a minimal block size $n_{\min}$, and $n = |I|$. Then the required storage for M, $N_{\mathcal{H},st}(M)$, is bounded by

$$N_{\mathcal{H},st}(M) \le 2C_{sp} \max\{k, n_{\min}\} (\operatorname{depth}(T_{I\times I}) + 1)n \in \mathcal{O}(C_{sp} kn \log_2 n) .$$

The storage complexity depends on the matrix size n, on the sparsity constant C_{sp}, on the minimal block size $n_{\min}$, and on the block-wise rank k. A complexity of the form

$$\mathcal{O}\left(C_{sp}C_{id}k^{\alpha}n\,(\log_2 n)^{\beta}\right),$$

with α and β independent of n will be called *almost linear* or *linear-polylogarithmic.*

Construction of $\mathcal{H}$-Matrices First, we have to construct the $\mathcal{H}$-tree, by, for example, cardinality or geometrically balanced clustering. Second, we use the admissibility condition to construct an $\mathcal{H}_\times$-tree. Now we have the structure for our $\mathcal{H}$-matrix. If the matrix M is given in a dense or sparse matrix format, then we can simply truncate M in the admissible leaves so that the rank condition is fulfilled. In the inadmissible leaves we copy M.

This approach has the main drawback that giving M in the dense format requires $\mathcal{O}(n^2)$ storage entries. Alternatively, we often can access M only element-wise, since the entry M_{ij} is computed using a quadrature formula like in the BEM example, but only in the moment it is required. In the inadmissible leaves we again need all entries, but in the admissible leaves we can, however, use sophisticated techniques. This techniques compute the approximation by using only a few entries so that constructing the $\mathcal{H}$-matrix can be done in linear-polylogarithmic complexity. Examples of this techniques are adaptive cross approximation [5] and hybrid cross approximation [20], which are based on the incomplete cross approximation [99].

2.3.3 Hierarchical Arithmetic

In the last lemma we have seen that $\mathcal{H}$-matrices can be stored in a cheap and efficient way, with an almost linear amount of storage (in the matrix dimension n). Now we will

Algorithm 2.1: $\mathcal{H}$-matrix-vector multiplication.

```
Input: $M \in \mathcal{H}(T_{I\times I})$, $v \in \mathbb{R}^n$
Output: $Mv = w \in \mathbb{R}^n$
1 $w := 0$;
2 forall $r \times s \in \mathcal{L}(T_{I\times I})$ do
3     $w_r = w_r + M_{r\times s} v_s$;
4 end
```

review some of the existing arithmetic operations for hierarchical matrices. There are two different approaches of arithmetic operations for $\mathcal{H}$-matrices; one fixes the block-wise rank of the intermediate along with the final results and the other fixes the approximation accuracy.

$\mathcal{H}$-Matrix-Vector Product The $\mathcal{H}$-matrix M has the following block structure:

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

If one partitions $v = \begin{bmatrix} v_1^T & v_2^T \end{bmatrix}^T$ in the same row way, then one can split the $\mathcal{H}$-matrix-vector product into four smaller $\mathcal{H}$-matrix-vector-products

$$Mv = \begin{bmatrix} M_{11}v_1 + M_{12}v_2 \\ M_{21}v_1 + M_{22}v_2 \end{bmatrix}.$$

We can repeat this process down to the leaves through recursion. In the leaves we have to form $w_r = w_r + M|_{(r,s)} v_s$. If (r, s) is inadmissible, then this is a standard matrix vector product. If (r, s) is admissible then we can use the low rank factorization $w_r = w_r + AB^T v_s$ to reduce the complexity of this operation.

Lemma 2.3.10: [49, Lemma 2.5]
Let $M \in \mathcal{H}(T_{I\times I}, k)$ be an $\mathcal{H}$-matrix with sparsity constant C_{sp}, minimal block size $n_{\min}$, and $n = |I|$. Then

$$N_{\mathcal{H}\cdot v}(M) \leq N_{\mathcal{H},st}(M) \in \mathcal{O}\left(C_{sp} k n \log_2 n\right). \tag{2.18}$$

Proof. During this process we have to access each storage entry of M once. ■

$\mathcal{H}$-Matrix-Matrix Addition The exact addition of $M, N \in \mathcal{H}(T, k)$ is trivial. Since M and N have the same hierarchical structure, it is sufficient to add the submatrices in the leaves. In the inadmissible leaves we have to add two dense matrices.

$$M +_{\mathcal{H}} N = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix} + \begin{bmatrix} N_{11} & N_{12} \\ N_{21} & N_{22} \end{bmatrix} = \begin{bmatrix} M_{11} +_{\mathcal{H}} N_{11} & M_{12} +_{\mathcal{H}} N_{12} \\ M_{21} +_{\mathcal{H}} N_{21} & M_{22} +_{\mathcal{H}} N_{22} \end{bmatrix}$$

However, in the admissible leaves we add two low rank matrices of rank at most k. In general, this leads to a low rank matrix of rank at most $2k$. This means that in general $M + N \in \mathcal{H}(T, 2k)$. This exact addition costs $N_{\mathcal{H}+\mathcal{H}} = N_{\mathcal{H},st}(M) + N_{\mathcal{H},st}(N)$.

One often requires only an approximate addition of two matrices that already contain an error like discretizations. In this case it is desirable to get a result with block-wise rank k, too. One should therefore perform a truncation step after the addition, which will reduce the ranks. The truncation increases the cost of one addition, but further additions with the resulting matrix are cheaper, since the block-wise ranks becomes smaller. The truncation is described in the next paragraph. The approximate addition is defined by

$$\mathcal{H}(T, k) \times \mathcal{H}(T, k) \to \mathcal{H}(T, k) : M +_{\mathcal{H}} N = \mathcal{T}_{k \leftarrow 2k}(M + N).$$

Sometimes one requires the addition of two $\mathcal{H}$-matrices that are not based on the same tree or the same admissibility condition. In this case one has to do a truncation of the matrices M and N to a common hierarchical structure first, see [48, 56]. This common structure is often the one of M or N depending on which one is finer.

If $M \in \mathcal{H}(k_1)$, $N \in \mathcal{H}(k_2)$, and $M +_{\mathcal{H}} N \in \mathcal{H}(k)$, with $k < k_1 + k_2$, then the truncated addition costs ([56, Lemma 7.8.4]) are

$$N_{\mathcal{H}+\mathcal{H}} \leq 6(k_1 + k_2) N_{\mathcal{H},st}(M + N) + 44(k_1 + k_2)^3 C_{sp} n.$$

For $k = k_1 = k_2$ we get

$$N_{\mathcal{H}+\mathcal{H}} \in \mathcal{O}(C_{sp} k^2 n \log_2 n + C_{sp} k^3 n).$$

Truncation of an $\mathcal{H}$-Matrix In many arithmetic operations it is necessary to transform an $\mathcal{H}$-matrix $M \in \mathcal{H}(T, \mathcal{A}, k)$ into an $\mathcal{H}(T', \mathcal{A}', k')$-matrix. We define

$$\mathcal{T}_{(T',\mathcal{A}',k') \leftarrow (T,\mathcal{A},k)} : \mathcal{H}(T, \mathcal{A}, k) \to \mathcal{H}(T', \mathcal{A}', k').$$

Some of these transformations, like $\mathcal{T}_{(T,\mathcal{A},2k) \leftarrow (T,\mathcal{A},k)}$, are trivial.

The most important truncation is $\mathcal{T}_{k' \leftarrow k} := \mathcal{T}_{(T,\mathcal{A},k') \leftarrow (T,\mathcal{A},k)}$, $k' < k$, which has the same tree T and the same admissibility condition $\mathcal{A}$. This truncation is processed block by block in the admissible blocks while the inadmissible blocks stay unchanged.

Let AB^T be the factorization of an admissible block $M|_b$ with rank k. First compute the QR factorizations $A = Q_A R_A$ and $B = Q_B R_B$, then compute the SVD $R_A R_B^T = U \Sigma V^T$. The singular values in Σ are ordered in decreasing order. Reducing the rank to k' means setting all singular values except the first k' values to zero. Approximating the block with a block of lowest rank with a maximal approximation error of ϵ' in the 2-norm means setting all singular values smaller than ϵ' to zero. In the Frobenius norm we have to set $\sigma_{k'+1}, \ldots, \sigma_k$ to zero, so that the sum $\sigma_{k'+1}^2 + \ldots + \sigma_k^2 \leq \epsilon'$. In the case of approximation to a given accuracy, we have to distribute the given maximal error ϵ for the whole matrix to the single blocks.

The truncation of an $\mathcal{H}$-matrix $M \in \mathcal{H}(T, k)$ to an $\mathcal{H}(T, k')$-matrix can be computed with complexity

$$\begin{aligned} N_{\mathcal{H},k'\leftarrow k} &\leq 12 C_{sp} \max\left\{k, n_{\min}\right\}^2 (\operatorname{depth}(T_{I\times I}) + 1) n + 46 C_{sp} k^3 n \\ &\in \mathcal{O}\left(C_{sp} k^2 n \log_2 n + C_{sp} k^3 n\right). \end{aligned}$$

This has been proved in [49, Lemma 2.2/2.9].

The other truncations consists of two steps; first bring the matrix to the target block-structure and then reduce the ranks by the truncation described above.

$\mathcal{H}$-Matrix-Matrix Product The product between M and N is also done by recursion, since

$$\begin{aligned} M *_{\mathcal{H}} N &= \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix} *_{\mathcal{H}} \begin{bmatrix} N_{11} & N_{12} \\ N_{21} & N_{22} \end{bmatrix} \\ &= \begin{bmatrix} M_{11} *_{\mathcal{H}} N_{11} +_{\mathcal{H}} M_{12} *_{\mathcal{H}} N_{21} & M_{11} *_{\mathcal{H}} N_{12} +_{\mathcal{H}} M_{12} *_{\mathcal{H}} N_{22} \\ M_{21} *_{\mathcal{H}} N_{11} +_{\mathcal{H}} M_{22} *_{\mathcal{H}} N_{21} & M_{21} *_{\mathcal{H}} N_{12} +_{\mathcal{H}} M_{22} *_{\mathcal{H}} N_{22} \end{bmatrix}. \end{aligned}$$

The computations in the leaves are more complicated, especially since some blocks have to be divided resp. combined to do the multiplication.

Again, truncation is necessary. Depending on whether we use truncation for the intermediate results or not, we get $\mathcal{H} * \mathcal{H}$ or $\mathcal{H} * \mathcal{H}$, best. The second one is of course more expensive. The $\mathcal{H}$-matrix-matrix product costs ([56, Lemma 7.8.21])

$$N_{\mathcal{H}*\mathcal{H}} \in \mathcal{O}\left(C_{sp}^2 \max\{C_{sp}, C_{id}\} k n \left(\log_2 n\right)^2 + C_{sp} C_{id} k^3 n \log_2 n\right),$$

and in best approximation

$$N_{\mathcal{H}*\mathcal{H},\text{best}} \in \mathcal{O}\left(C_{sp}^3 C_{id}^3 k^3 n \left(\log_2 n\right)^3\right).$$

$\mathcal{H}$-Cholesky/LU Decomposition The $\mathcal{H}$-Cholesky decomposition in the symmetric case and the $\mathcal{H}$-LU decomposition, in general, are approximate triangular factorizations, which can be used as preconditioners for the solutions of linear equations. Both factorizations decompose M in a lower and upper triangular matrix. Up to this point we have done all the definitions without introducing an ordering of the indices in I. Such an ordering is thus required to define the lower/upper triangular.

The factorizations are done in a block recursive way, so the ordering should be chosen with respect to this. We use the following recursive mapping from I to $\{1, \ldots, n\}$:

(i) we number $\mathrm{root}(T_I) = I$ with indices out of $\{1, \ldots, n\}$,

(ii) let $v \in T_I$ have the new indices $\left\{v_1, \ldots, v_{|v|}\right\}$, then the sons s_1 and $s_2 \in S(v)$ get the indices $\left\{v_1, \ldots, v_{|s_1|}\right\}$ and $\left\{v_{|s_1|+1}, \ldots, v_{|v|}\right\}$, and

(iii) number the indices in $v \in \mathcal{L}(T_I)$ arbitrary.

Definition 2.3.11: (lower/upper triangular $\mathcal{H}$-matrix) For two blocks r and s with $r \cap s = \emptyset$, we say $r < s$ if $\forall i \in r, j \in s : i < j$. For an *upper triangular $\mathcal{H}$-matrix* all blocks $b = (r, s)$ with $s < r$ are zero and all leaves $b = (r, r)$ are inadmissible blocks, with an upper triangular dense matrix inside. A *lower triangular $\mathcal{H}$-matrix* is the transpose of an upper triangular one.

Let M be an $\mathcal{H}(T, k)$-matrix with an ordering of the indices, as described above. Then we can compute the LU decomposition by the following block recursion:

$$\begin{aligned} M = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix} &= \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix} \\ \Leftrightarrow M_{11} = L_{11}U_{11}, &\quad M_{12} = L_{11}U_{12} \\ M_{21} = L_{21}U_{11}, &\quad M_{22} = L_{21}U_{12} + L_{22}U_{22}. \end{aligned}$$

For the computations of U_{12} and L_{21} we have to solve a triangular system. The computations of L_{11}, U_{11}, L_{22} and U_{22} are done by recursion. The recursion is working only on the diagonal. We should assume that the leaves on the diagonal are all inadmissible, so that we have only to do dense LU factorizations for dense matrices.

The $\mathcal{H}$-LU factorization and the $\mathcal{H}$-Cholesky decomposition have the same complexity as the $\mathcal{H}$-matrix-matrix multiplication, see [56, Lemma 7.8.23].

The hierarchical block structure does not permit permutation of indices, so that we are not able to do pivoting. This may lead to large errors in the LU decomposition. The Cholesky decomposition for hierarchical matrices is analog, see Algorithm 2.2. Since the Cholesky decomposition is computed only for symmetric positive definite matrices, accuracy is guaranteed without pivoting. In Subsection 6.2.3 a two-step version of the Cholesky decomposition is discussed, which allows for a way to choose the $\mathcal{H}$-matrix accuracy so that a given accuracy in the Cholesky decomposition is achieved.

$\mathcal{H}$-Inversion The inversion of hierarchical matrices is already treated in the first $\mathcal{H}$-matrix paper [55]. In many subsequent publications the $\mathcal{H}$-inversion is a subject of constant investigation, in [48, 73, 21, 49, 58, 6, 56] among others.

The inverse of an $\mathcal{H}$-matrix M can be computed through a recursive block Gaussian elimination [55], a Schulz iteration [48], or multiplying the inverses of the LU factors of M [4]. The Schulz iteration [94] computes the inverse by solving

$$MX = I$$

Algorithm 2.2: $\mathcal{H}$-Cholesky Decomposition [56].

```
Input: $M \in \mathcal{H}(T_{I\times I})$, $r = I$
Output: $L \in \mathcal{H}(T_{I\times I})$, with $LL^T = M$
1  Function $\mathcal{H}$-Cholesky factorization($M|_{r\times r}$, $r$)
2  if $r \times r \in \mathcal{L}(T_{I\times I})$ then
3      $L|_{r\times r}$ :=Cholesky factorization($M|_{r\times r}$);    /* $M|_{r\times r}$ is dense, use
       standard Cholesky factorization */
4  else
5      Assume $\{s_1, s_2\} = S(r)$;                  /* $T_I$ is a binary tree */
6      $L|_{s_1\times s_1}$ :=$\mathcal{H}$-Cholesky factorization($M|_{s_1\times s_1}$, $s_1$);
7      Compute $L|_{s_2\times s_1}$, so that $L|_{s_2\times s_1} L|_{s_1\times s_1} = M|_{s_2\times s_1}$;
8      $L|_{s_1\times s_1}$ :=$\mathcal{H}$-Cholesky factorization($M|_{s_2\times s_2} - L|_{s_2\times s_1} L|^T_{s_2\times s_1}$, $s_2$);
9  end
10 return $L_{r\times r}$;
```

by a Newton method. This is inefficient, as many iterations are required and a truncation is necessary in each iteration.

The block Gaussian elimination seems to be the method of choice, so in Subsection 6.2.2 the two-step version of this inversion [48] is used.

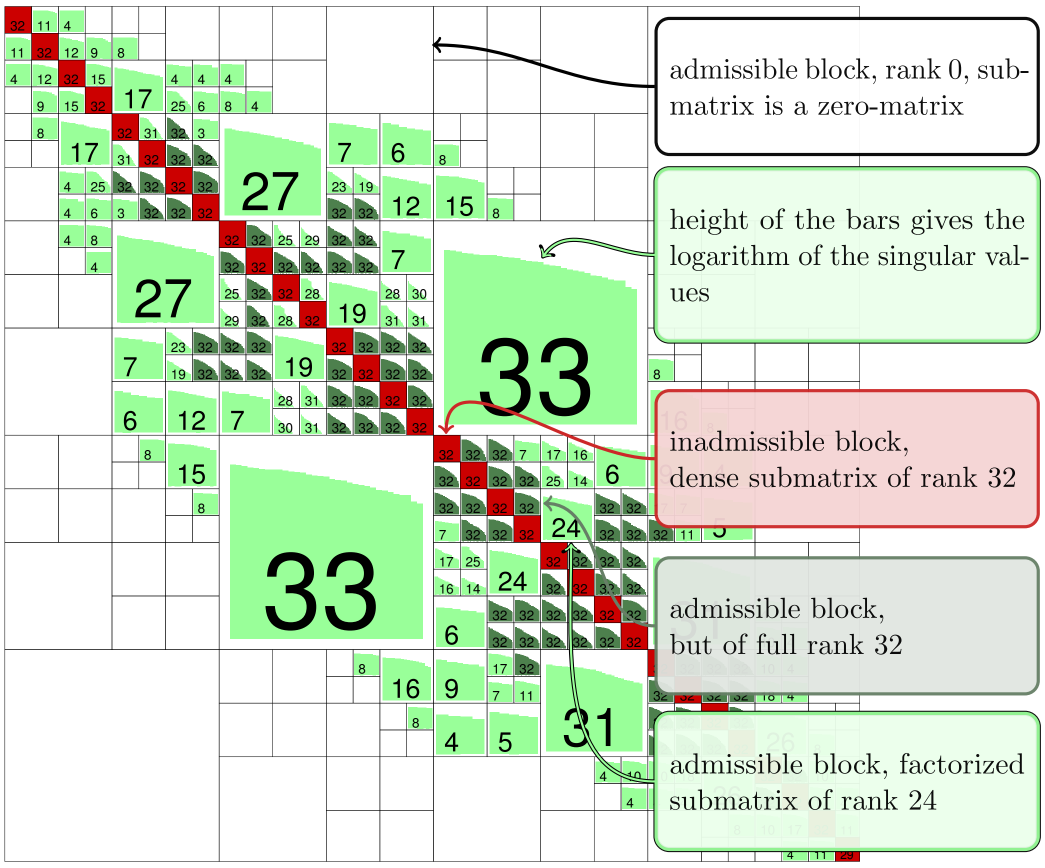
The $\mathcal{H}$-inversion is also as expensive as the $\mathcal{H}$-matrix-matrix multiplication, see [56, Lemma 7.8.22].

All these arithmetic operations are of linear-polylogarithmic complexity. These operations (and a lot more) are implemented in the $\mathcal{H}$Lib [65], which is used for the numerical computations in the subsequent chapters.

$\mathcal{H}$-Matrix Structure Diagrams The $\mathcal{H}$Lib provides a visualization function for the structure of an $\mathcal{H}$-matrix. We will have a look at such diagrams in Chapter 4, as such it is appropriate that they should also be explained here. In Figure 2.7 the structure of an $\mathcal{H}$-matrix is shown.

The rectangles represent the submatrices of the matrix as well as the nodes of the $\mathcal{H}$-block tree $T_{I\times I}$. Green rectangles stand for admissible blocks resp. factorized submatrices. Red rectangles stand for inadmissible blocks resp. dense submatrices. The dark green blocks are admissible blocks like the green ones, but have become full rank by intermediate computations. The rank of the factorization of a submatrix is given by the number in the rectangle. The number in the red rectangles give the size of the block. In the admissible blocks the green bars show the logarithmic value of the singular values of the submatrix. Empty rectangles stand for admissible blocks, where the corresponding submatrices are zero, meaning that the rank is also 0.

In the next subsection $\mathcal{H}_\ell$-matrices are investigated. Afterwards we have a look at some



admissible block, rank 0, submatrix is a zero-matrix
height of the bars gives the logarithm of the singular values
inadmissible block, dense submatrix of rank 32
admissible block, but of full rank 32
admissible block, factorized submatrix of rank 24

Figure 2.7: $\mathcal{H}$-matrix structure diagram.

examples of $\mathcal{H}$-matrices, which will be used for the numerical examples.

2.3.4 Simple Hierarchical Matrices ($\mathcal{H}_\ell$-Matrices)

Already in the first paper on hierachical matrices [55] the subset of simple hierarchical matrices was mentioned. In [56] these matrices are called $\mathcal{H}_p$-matrices, with p the depth of the hierarchical structure. The following definition is slightly different, since we stop the subdivision for blocks smaller than $n_{\min}$ like for $\mathcal{H}$-matrices, this increases the storage efficiency and the efficiency of the arithmetic operations. Here they are called $\mathcal{H}_\ell$-matrices, with $\ell = \operatorname{depth}(T)$ as in [47].

Definition 2.3.12: ($\mathcal{H}_\ell$-matrix)
Let $I = \{1, \ldots, n\}$ be an index set and $n = 2^\ell n_0$ with $\ell \in \mathbb{N}$. A matrix $M \in \mathbb{R}^{I \times I}$ is called an $\mathcal{H}_\ell$-matrix of block-wise rank k, $M \in \mathcal{H}_\ell(k)$ for short, if it fulfills the following recursive conditions:

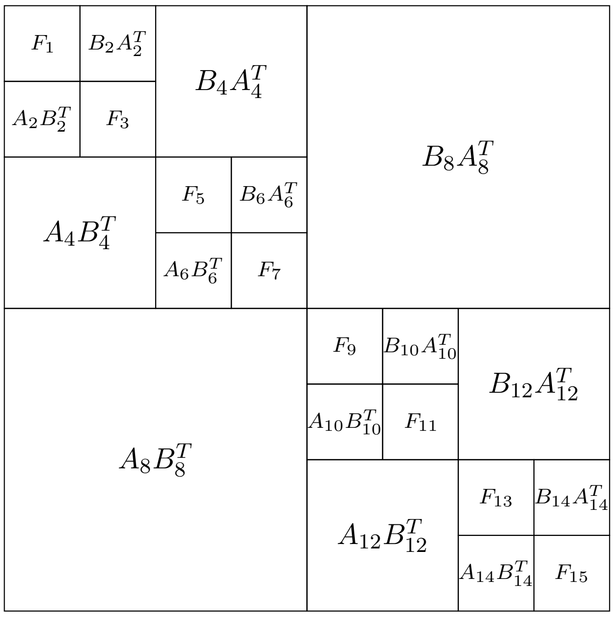


Figure 2.8: Structure of an $\mathcal{H}_3$-matrix.

1. $\ell = 0$: $n_0 \leq n_{\min}$, $M \in \mathcal{H}_0(k)$ if $M \in \mathbb{R}^{n_0 \times n_0}$ and
2. $n_\ell = 2^\ell n_0$: M is partitioned in

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

with $M_{11}, M_{22} \in \mathcal{H}_{\ell-1}(k)$, $M_{12} = A_1 B_1^T$ and $M_{21} = B_2 A_2^T$, where $A_i, B_i \in \mathbb{R}^{n_{\ell-1} \times k'}$, with $k' \leq k$.

We are interested only in symmetric $\mathcal{H}_\ell$-matrices, so we have $M_{12} = M_{21}^T$, $A_1 = A_2$ and $B_1 = B_2$. A symmetric $\mathcal{H}_3$-matrix is depicted in Figure 2.8.

Lemma 2.3.13: If $M \in \mathcal{H}_\ell(k)$, then M is a hierarchical matrix of block-wise rank k, too.

Proof. The minimum block size is $n_0 \leq n_{\min}$. The $\mathcal{H}$-tree T_I is a binary tree, which divides each node $r = \{i_1, \ldots, i_m\}$ into $r_1 = \left\{i_1, \ldots, i_{m/2}\right\}$ and $r_2 = \left\{i_{m/2+1}, \ldots, i_m\right\}$ on the next level. In the $\mathcal{H}_\times$-tree only nodes of the type $r \times r$ are subdivided. The other nodes $r \times s$, with $r \cap s = \emptyset$, correspond to blocks M_{12} or M_{21}, which have at most rank k.

■

Remark 2.3.14: The $\mathcal{H}_\ell$-matrices are $\mathcal{H}$-matrices with the weak admissibility condition $\mathcal{A}_{\text{weak}}$, see Equation (2.13), and binary, cardinality based clustering. These $\mathcal{H}$-matrices with weak admissibility condition are investigated in [58].

Corollary 2.3.15: Let $M \in \mathcal{H}_\ell(k)$. Then the sparsity constant of M is $C_{sp} = 2$ and the idempotency constant is $C_{id} = 1$.

Corollary 2.3.16: Let $M \in \mathcal{H}_\ell(k)$. Then $N_{\mathcal{H}_\ell,st}(M) = 2kn \log_2 n + nn_{\min}$.

The following remark shows that $\mathcal{H}_\ell$-matrices originate typically from one-dimensional problems.

Remark 2.3.17: If $M \in \mathcal{H}(T_{I\times I}, k)$ origins from the discretization of a one-dimensional problem with standard admissibility condition and $\ell = \operatorname{depth}(T_{I\times I})$, then M is also an $\mathcal{H}$-matrix with weak admissibility condition and block-wise rank $k' < \ell k$, $M \in \mathcal{H}_\ell(k')$, see again [58].

The following is a further inclusion:

Lemma 2.3.18: Let $T \in \mathbb{R}^{2^\ell n_0 \times 2^\ell n_0}$ be a tridiagonal matrix, $T_{ij} = 0 \ \forall i, j$ with $|i - j| > 1$. Then $T \in \mathcal{H}_\ell(1) \subset \mathcal{H}(1)$.

This lemma is used for Figure 2.11. Tridiagonal matrices are well studied. Here there are several good eigenvalue algorithms for tridiagonal matrices, e.g., dsqd algorithm [40] or the MR^3 algorithm [108]. Due to this inclusion, we should not expect to find a faster eigenvalue algorithm for $\mathcal{H}_\ell$-matrices than for tridiagonal matrices. The best known eigenvalue algorithms for symmetric tridiagonal matrices have quadratic complexity.

The structure of $\mathcal{H}_\ell$-matrices is poorer than the structure of $\mathcal{H}$-matrices, but the arithmetic operations are cheaper. For instance, there is an exact inversion.

Lemma 2.3.19: Let $M \in \mathcal{H}_\ell(k)$ be invertible. Then $M^{-1} \in \mathcal{H}_\ell(k\ell)$.

This is a loose version of [58, Corollary 4.7]. To prove this we require the following definition.

Definition 2.3.20: [58, Definition 4.1]
A matrix M belongs to the set $\mathcal{M}_{k,\tau}$, if

$$\operatorname{rank}\left(M|_{\tau'\times\tau}\right) \le k \quad \text{and} \quad \operatorname{rank}\left(M|_{\tau\times\tau'}\right) \le k, \tag{2.19}$$

with $\tau' = I \setminus \tau$.

Proof. (Lemma 2.3.19)
This proof is based on the argumentation leading to [58, Corollary 4.7].

We have $M \in \mathcal{M}_{kp,\tau}$ for all $\tau \in T_I^{(p)}$, since $M \in \mathcal{H}_\ell(k)$ and $\tau \times I \setminus \tau$ have intersections with p blocks of rank at most k, see [58, Lemma 4.5]. So we have $M \in \mathcal{M}_{k\ell,\tau}$ for all $\tau \in T_I$, since depth $(T_I) = \ell$.

In [58, Lemma 4.2] it is shown that if $M \in \mathcal{M}_{k,\tau}$, then $M^{-1} \in \mathcal{M}_{k,\tau}$. The proof uses the inversion by Schur complement with $S = M|_{\tau'\times\tau'} - M|_{\tau'\times\tau}\, M|_{\tau\times\tau}^{-1}\, M|_{\tau\times\tau'}$, so that

$$\operatorname{rank}\left(M^{-1}\big|_{\tau'\times\tau}\right) = \operatorname{rank}\left(-S^{-1}\, M|_{\tau'\times\tau}\, M^{-1}\big|_{\tau\times\tau}\right) \leq \operatorname{rank}\left(M|_{\tau'\times\tau}\right) \leq k.$$

If $M|_{\tau\times\tau}$ is not invertible, one has to introduce a small perturbation to make the block invertible.

Obviously if for all $\tau \in T_I : M \in \mathcal{M}_{k\ell,\tau}$, then $M \in \mathcal{H}_\ell(k\ell)$, see [58, Remark 4.4].

■

In Chapter 5 we will investigate an eigenvalue algorithm that performs especially well for $\mathcal{H}_\ell$-matrices. Further we will show in Chapter 5 that there is an efficient and exact LDL^T decomposition for $M \in \mathcal{H}_\ell(k)$. In the next section some series of example matrices, which will be used for testing the algorithms, will be defined.

2.4 Examples

The following examples are used to measure and compare the quality of the (eigenvalue) algorithms in the subsequent chapters.

2.4.1 FEM Example

As we have seen in Example 2.3.1, the inverse of

$$\Delta_{2,h} = I \otimes \Delta_h + \Delta_h \otimes I \tag{2.20}$$

has a good approximation by $\mathcal{H}$-matrices. The matrix itself is sparse with, at most 5 non-zero entries per row and can be represented as an $\mathcal{H}$-matrix. Therefore, one has to partition the index set, that follows the geometrically balanced clustering. This leads to a matrix with a larger bandwidth then the standard indexing as suggested in Equation (2.20). However, the inverse has a better approximation by an $\mathcal{H}$-matrix.

Lemma 2.4.1: The finite difference method (FDM) discretization of the 2D Laplace problem

$$\Delta f(x) = 0 \quad \text{for } x \in [0,1]^2\,, \tag{2.21}$$

with n inner uniform distributed discretization points gives the matrix $\Delta_{2,h}$.

This example is part of the $\mathcal{H}$Lib [65]. We choose $n_{\min} = 32$.

For the 3D Laplace problem we get analogously

$$\Delta_{3,h} = I \otimes I \otimes \Delta_h + I \otimes \Delta_h \otimes I + \Delta_h \otimes I \otimes I$$

as the FDM discretization of

$$\Delta f(x) = 0 \quad \text{for } x \in [0,1]^3 .$$

Lemma 2.4.2: The eigenpairs of $\Delta_{2,h} \in \mathbb{R}^{n^2 \times n^2}$ are $(\lambda_i + \mu_j, u_i \otimes v_j)$, where (λ_i, u_i) and (μ_j, v_j) are eigenpairs of Δ_h. Analogously for $\Delta_{3,h} \in \mathbb{R}^{n^3 \times n^3}$ the eigenpairs are $(\lambda_i + \mu_j + \nu_k, u_i \otimes v_j \otimes w_k)$, where (λ_i, u_i), (μ_j, v_j) and (ν_k, w_k) are eigenpairs of Δ_h.

Proof. We have

$$\begin{aligned} \Delta_{2,h}(u_i \otimes v_j) &= (I \otimes \Delta_h)(u_i \otimes v_j) + (\Delta_h \otimes I)(u_i \otimes v_j) \\ &= u_i \otimes (\Delta_h v_j) + (\Delta_h u_i) \otimes v_j = \mu_j(u_i \otimes v_j) + \lambda_i(u_i \otimes v_j). \end{aligned}$$

There are n^2 sums $\lambda_i + \mu_j$, since $i, j \in \{1, \ldots, n\}$. So we have found all n^2 eigenvalues of $\Delta_{2,h}$. Analog for $\Delta_{3,h}$.

■

The eigenvalues of $\Delta_{2,h}$ are

$$4\left(\sin\left(\frac{i\pi}{2(n+1)}\right)\right)^2 + 4\left(\sin\left(\frac{j\pi}{2(n+1)}\right)\right)^2 \quad \forall i, j \in \{1, \ldots, n\}$$

and the eigenvalues of $\Delta_{3,h}$ are

$$4\sum_{j=1}^{3}\left(\sin\left(\frac{i_j\pi}{2(n+1)}\right)\right)^2 \quad \forall i_1, i_2, i_3 \in \{1, \ldots, n\}.$$

For the test, we use the 2D FEM matrices with 4 to 2 048 inner discretization points and 3D FEM matrices with 4 to 64 inner discretization points. The properties of these matrices are listed in Tables 2.1 and 2.2. Further, the matrices are the $\mathcal{H}$-matrix representations of sparse matrices. This means they have large blocks of rank 0. The low rank factorization of the non-zero blocks also leads to sparse matrices. They are not, however, stored as sparse matrices.

In Figure 2.9 and 2.10 the smallest eigenvectors of these matrices are shown.

Name	n	C_{sp}	C_{id}	Condition κ	$N_{\mathcal{H},st}$ in kB
FEM8	64	2	1	3.2163 e+01	21
FEM16	256	6	5	1.1646 e+02	101
FEM32	1 024	10	13	4.4069 e+02	449
FEM64	4 096	14	13	1.7117 e+03	1 890
FEM128	16 384	14	13	6.7437 e+03	7 750
FEM256	65 536	14	13	2.6768 e+04	31 384
FEM512	262 144	18	13	1.0666 e+05	126 321
FEM1024	1 048 576	30	17	4.2580 e+05	506 872
FEM2048	4 194 304	20	17	1.7015 e+06	2 030 704

Table 2.1: 2D FEM example matrices and their properties.

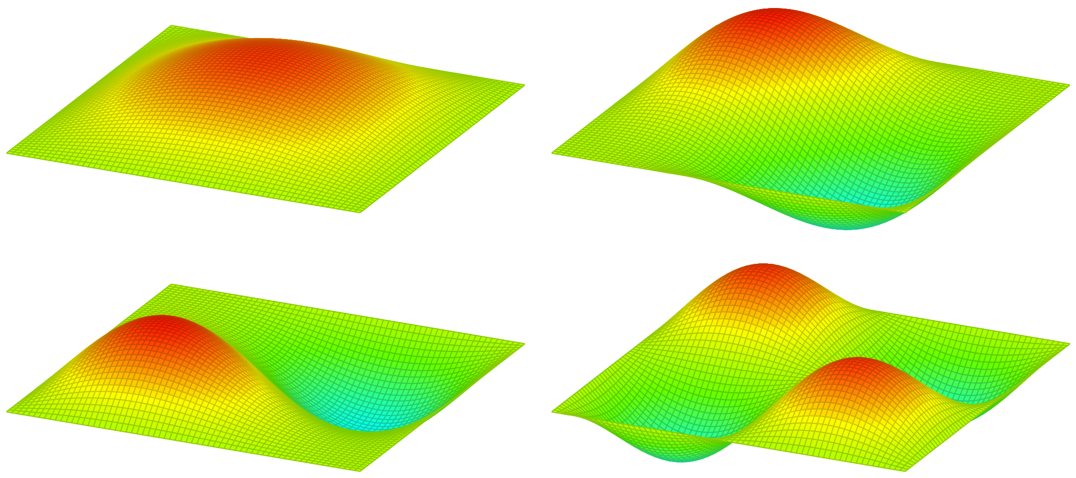


Figure 2.9: Eigenvectors of FEM32 corresponding to the four smallest eigenvalue ($\lambda_1, \lambda_2 = \lambda_3, \lambda_4$).

Name	n	C_{sp}	C_{id}	Condition κ	$N_{\mathcal{H},st}$ in kB
FEM3D4	64	2	1	9.4721 e+00	25
FEM3D8	512	8	9	3.2163 e+01	302
FEM3D12	1 728	32	129	6.7827 e+01	1 083
FEM3D16	4 096	20	257	1.1646 e+02	2 954
FEM3D32	32 768	32	257	4.4069 e+02	26 266
FEM3D64	262 144	148	449	1.7117 e+03	227 166

Table 2.2: 3D FEM example matrices and their properties.

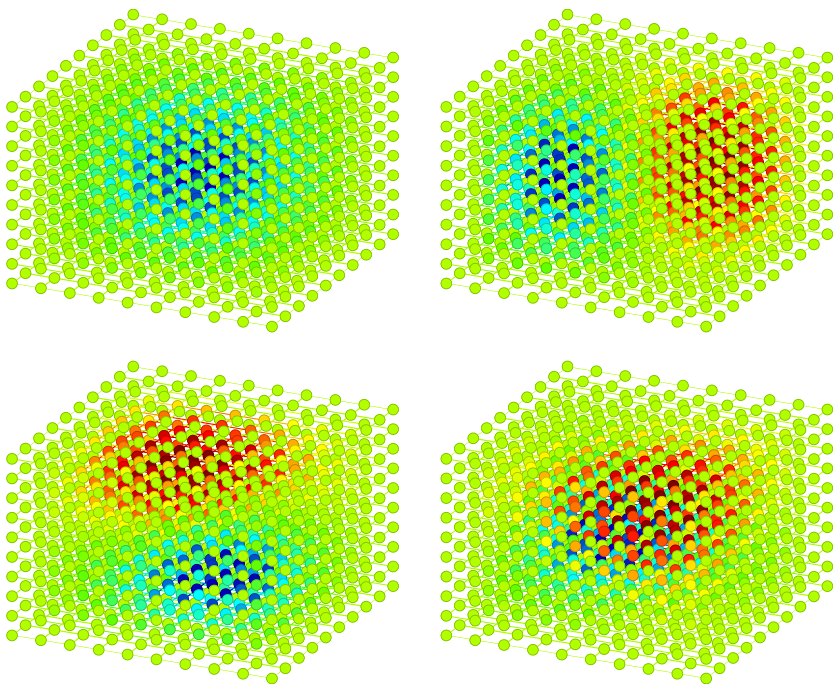


Figure 2.10: Eigenvectors of FEM3D8 corresponding to the four smallest eigenvalues ($\lambda_1, \lambda_2 = \lambda_3 = \lambda_4$).

Remark 2.4.3: For finite element problems the nested dissection (ND), a domain decomposition based method, is often superior to geometrically balanced clustering as shown in [51]. The author tested the usage of FEM matrices based on ND. Nested dissection is not useful for the LR Cholesky algorithm, since the computation in Table 4.1 would need more iterations and more time (about 40%) to compute approximations with larger errors compared with geometrically balanced clustering. Further, the author's implementation of the algorithm in Chapter 5 does not work for FEM matrices based on ND. On the other hand the usage of nested dissection would accelerate the computations in Table 6.1.

2.4.2 BEM Example

Beside the sparse FEM matrices, the algorithms should be tested with dense matrices. Therefore, we use the boundary element example from the $\mathcal{H}$Lib [65] with the single layer potential operator. The single layer potential operator is the one from Equation 2.6. We choose

$$\Omega = \left\{ x \in \mathbb{R}^3 \middle| \|x\|_2 \leq 1 \right\}$$

and hybrid cross approximation for the approximation in the admissible blocks. The eigenvalues of these matrices are small ($10^{-2} \ldots 10^{-9}$), so that we decide to normalize

Name	n	C_{sp}	C_{id}	Condition κ	$N_{\mathcal{H},st}$ in kB
BEM4	66	2	1	1.3730 e+02	45
BEM8	258	6	5	4.8990 e+02	450
BEM16	1 026	13	9	1.3903 e+03	2 991
BEM32	4 098	29	17	3.3556 e+03	19 239
BEM64	16 386	53	41	7.4343 e+03	112 723
BEM128	65 538	61	100	*1.5884 e+04*	662 141

Table 2.3: BEM example matrices and their properties.

the matrices to $\|M\|_2 = 1$. We use matrices of dimension 66 to 65 538 with minimal block size $n_{\min} = 32$. The largest matrices is 662 MB large, which is fairly small compared with the 32 GB storage needed when storing in the dense matrix format.

Since the kernel function has an unbounded support, the matrices are dense. The other properties of the matrices are shown in Table 2.3. The italic condition numbers are computed using the algorithm from Chapter 5.

2.4.3 Randomly Generated Examples

The $\mathcal{H}_\ell$-matrices are a subset of the $\mathcal{H}$-matrices and have a simple structure, see Corollary 2.3.15. We will test some of our algorithms with these simple structured matrices, too. Therefore, we generate the structure with $n = 2^\ell\, n_{\min}$, $n_{\min} = 32$ and fill the admissible blocks in the lower triangle with randomly-generated, low-rank factors of rank $k = 1, 2, \ldots, 16$. The upper triangular part is the transposed copy of the lower triangle to ensure symmetry. The matrices are shifted to become positive definite. Finally, the matrices are normalized to a spectral norm of 1. We name these matrices with Hℓ r1 ($\ell = 1, 2, \ldots, 15$) and H9 rk ($k = 1, 2, 3, 4, 8, 16$). Here we pick an arbitrary value for the depth $\ell = 9$ to generate examples with different block-wise rank. H1 r1 has the size 64×64 and H15 r1 has the size $1\,048\,576 \times 1\,048\,576$. The properties of these matrices are listed in Table 2.4. The condition numbers of the matrices Hℓ r1, $\ell = 1, 2, \ldots, 10$, are computed via the smallest and largest eigenvalues computed by the LAPACK eigensolver dsyev. For the condition numbers of the larger matrices, we use the slicing algorithm described in Chapter 5 and write the condition numbers in italics.

The HSS matrices, see Subsection 2.5.3, have even more structure than the $\mathcal{H}_\ell$-matrices. In Chapter 5, randomly generated examples, with respect to the common row and column spaces, are used. These matrices are normalized and are called HSSℓ r1, with size $n = 2^\ell\, n_{\min}$, $n_{\min} = 32$.

Name	n	C_{sp}	C_{id}	Condition κ	$N_{\mathcal{H},st}$ in kB
H1 r1	64	2	1	2.9136 e+00	18
H2 r1	128	2	1	5.1427 e+00	41
H3 r1	256	2	1	9.0674 e+00	103
H4 r1	512	2	1	1.7764 e+01	263
H5 r1	1 024	2	1	3.4028 e+01	670
H6 r1	2 048	2	1	7.1787 e+01	1 692
H7 r1	4 096	2	1	1.4036 e+02	4 213
H8 r1	8 192	2	1	2.7369 e+02	10 352
H9 r1	16 384	2	1	5.5160 e+02	25 056
H10 r1	32 768	2	1	1.0963 e+03	59 840
H11 r1	65 536	2	1	*2.2181 e+03*	107 392
H12 r1	131 072	2	1	*4.5436 e+03*	231 168
H13 r1	262 144	2	1	*9.2921 e+03*	495 104
H14 r1	524 288	2	1	*1.9539 e+04*	1 055 744
H15 r1	1 048 576	2	1	*3.7755 e+04*	2 242 560

Table 2.4: $\mathcal{H}_\ell(1)$ example matrices and their properties.

2.4.4 Application Based Examples

We will also test our algorithms using example matrices from applications. Therefore, we take four sparse matrices from the "University of Florida Sparse Matrix Collection", [33]. We use the matrices bcsstk08 ($n = 1\,074$, nnz $= 12\,960$), bcsstk38 ($n = 8\,032$, nnz $= 355\,460$), msc10848 ($n = 10\,848$, nnz $= 1\,229\,776$), and msc23052 ($n = 23\,052$, nnz $= 1\,142\,686$). These matrices originate from structural problems and are symmetric positive definite. We transform the sparse matrices into the $\mathcal{H}$-matrix format. We start with the whole matrix and partition it into four submatrices. If such a submatrix has less than 32 (bcsstk) or 1 024 (msc) non-zero entries and is not on the diagonal, then we store the submatrix as rank-k-matrix in factorized form. Otherwise we continue by recursion. We choose the minimal block size $n_{\min} = 32$ and finally we normalize the matrices to $\|M\|_2 = 1$.

2.4.5 One-Dimensional Integral Equation

The last test example is the integral equation eigenvalue problem

$$\int_\Gamma \log \|x - y\|_2 u(y) dy = \lambda u(x).$$

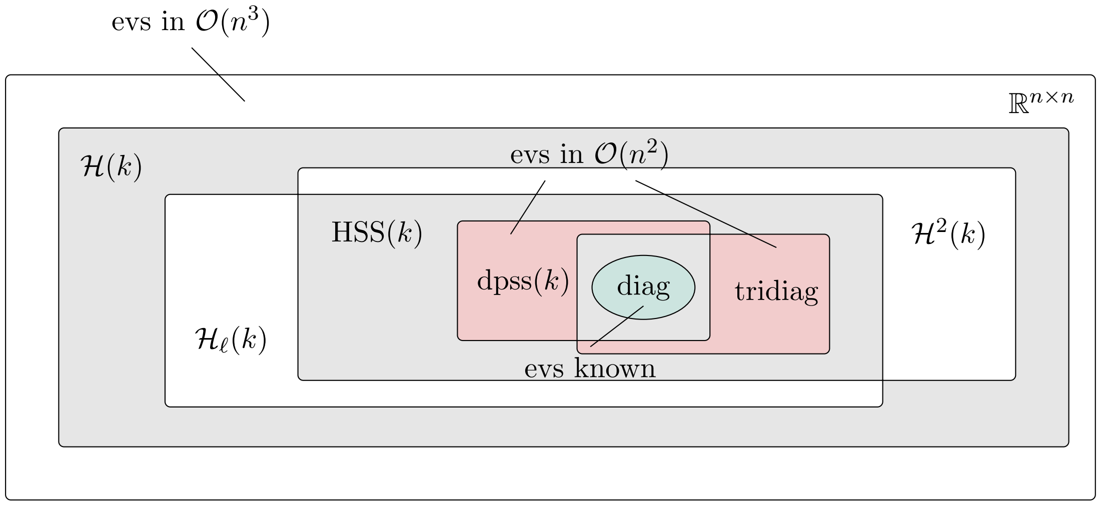


Figure 2.11: Set diagram of different matrix formats.

We choose the diagonal of the 2D-unit square for Γ. The variational formulation is

$$\sum_j \int_\Gamma \phi_i(x) \int_\Gamma \log \|x-y\|_2 \phi_j(y) dy c_j = \lambda \sum_j \int_\Gamma \phi_i(x)\phi_j(x) dx c_j = \lambda c_i.$$

We use piece-wise constant functions with $\int_\Gamma \phi_i(x)\phi_j(x)dx = \delta_{ij}$ for the discretization. The eigenvector to λ is then $(x_i)_{i=1}^n$. The kernel-function can be approximated by a piecewise separable function, see, e.g., [6]. Integral equations with the kernel function $\log \|x-y\|_2$ occur in the BEM discretization of the 2D Poisson problem.

The resulting matrices are symmetric but not positive definite. They are shifted and normalized so that all eigenvalues lie between 0 and 1. The minimal block size is chosen to be 32. We choose an approximation of the kernel function by interpolation of order 3 given in the $\mathcal{H}$Lib [65]. The matrices have the structure of $\mathcal{H}^2$-matrices (see next section) that we ignore. We use the weak admissible condition and get $\mathcal{H}_\ell$-matrices with block-wise rank 5 after a slight truncation. We use matrices from dimensions 64 to 16 384 and name them log64 to log16384.

In the next section the relations between the $\mathcal{H}$-matrices and other data-sparse matrix format are reviewed.

2.5 Related Matrix Formats

There are matrix formats that have some properties in common with the $\mathcal{H}$-matrix format. In this section a brief explanation of $\mathcal{H}^2$-, HSS, and semiseparable matrices is given. The inclusions of these matrix formats are shown in Figure 2.11.

2.5.1 $\mathcal{H}^2$-Matrices

The admissible blocks of a hierarchical matrix have all their own factorization AB^T. In some applications, such as the discretization of some integral equations, the As for the blocks in one block-row are in the same low-dimensional space. This is used in the $\mathcal{H}^2$-matrices to make the costs linear in n.

There is a row and a column *cluster basis*, which are defined as the families

$$V = (V_r)_{r\in T_I} \quad \text{and} \quad W = (W_s)_{s\in T_J}\,.$$

Further, we require a block $\mathcal{H}$-tree $T_{I\times J}$ and an admissibility condition $\mathcal{A}$ to define

$$\mathcal{H}^2(T_{I\times J},\mathcal{A},k,V,W) := \left\{ M\in\mathbb{R}^{I\times J} \;\middle|\; \begin{array}{l} \forall r\times s\in\mathcal{L}(T_{I\times J}):\\ \text{if } (\mathcal{A}(r,s)=\text{true})\,\text{then}\\ \;M_{r\times s}=V_rS_{r\times s}W_s^T,\;\text{with } S_{r\times s}\in\mathbb{R}^{k\times k},\\ \text{else}\\ \;\min\{|r|,|s|\}\le n_{\min} \end{array} \right\}.$$

A complete description of $\mathcal{H}^2$-matrices can be found in [18].

2.5.2 Diagonal plus Semiseparable Matrices

The inverses of symmetric tridiagonal matrices have a special structure. This was probably first observed by Gantmacher and Krein [100] and was the starting point for the investigation of semiseparable matrices. There are different definitions for semiseparable matrices. In the one given below we will use MATLAB notation for the lower/upper triangular part of a matrix. The functions $\text{tril}\,(M,j)$ and $\text{triu}\,(M,j)$ are defined by

$$\text{tril}\,(M,s):\mathbb{R}^{n\times n}\times\mathbb{Z}\to\mathbb{R}^{n\times n}:(\text{tril}\,(M,s))_{i,j}=\begin{cases}M_{i,j} & \text{if } i\ge j+s,\\ 0 & \text{else}\end{cases} \tag{2.22}$$

and

$$\text{triu}\,(M,s):\mathbb{R}^{n\times n}\times\mathbb{Z}\to\mathbb{R}^{n\times n}:(\text{triu}\,(M,s))_{i,j}=\begin{cases}M_{i,j} & \text{if } i\le j+s,\\ 0 & \text{else.}\end{cases} \tag{2.23}$$

If $s=0$, we omit s. The small example below explains the usage of $\text{tril}\,(M,s)$ and $\text{triu}\,(M,s)$.

Example 2.5.1: Let

$$M=\begin{bmatrix}1&2&3\\4&5&6\\7&8&9\end{bmatrix}.$$

Then we have

$$\operatorname{tril}(M) = \begin{bmatrix} 1 & 0 & 0 \\ 4 & 5 & 0 \\ 7 & 8 & 9 \end{bmatrix}, \qquad \operatorname{triu}(M) = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 5 & 6 \\ 0 & 0 & 9 \end{bmatrix},$$

$$\operatorname{tril}(M,-1) = \begin{bmatrix} 0 & 0 & 0 \\ 4 & 0 & 0 \\ 7 & 8 & 0 \end{bmatrix}, \quad \text{and} \qquad \operatorname{triu}(M,1) = \begin{bmatrix} 0 & 2 & 3 \\ 0 & 0 & 6 \\ 0 & 0 & 0 \end{bmatrix}.$$

In Chapter 4 we will use the definition of generator representable semiseparable matrices.

Definition 2.5.2: (diagonal plus semiseparable matrix)
Let $M = M^T \in \mathbb{R}^{n\times n}$. If M can be written (using MATLAB notation) in the form

$$M = \operatorname{diag}(d) + \sum_{i=1}^{r} \left(\operatorname{tril}\left(u^i v^{iT}\right) + \operatorname{triu}\left(v^i u^{iT}\right)\right), \tag{2.24}$$

with $d, u^i, v^i \in \mathbb{R}^n$, then M is a *symmetric (generator representable) diagonal plus semiseparable matrix of semiseparability rank* r.

Remark 2.5.3: The representation of M by $u^i, v^i, i = 1, \ldots, r$, is for small r a data-sparse representation, since $(2r + 1)n$ parameters are sufficient for the description of M. In Chapter 5 we will also use this representation for $r > n$. The representation is thus neither data-sparse nor efficient, but is nonetheless useful for our theoretical argumentation.

Remark 2.5.4: The discretization of semiseparable kernels,

$$k(x,y) = \begin{cases} g(x)f(y) & \text{if } x \leq y, \\ f(x)g(y) & \text{if } y \leq x, \end{cases}$$

leads to semiseparable matrices, see [103, Subsection 3.5.1].

Semiseparable matrices are a special case of $\mathcal{H}_\ell$-matrices as the following corollary shows.

Corollary 2.5.5: Let M be a generator representable diagonal plus semiseparable matrix of rank r and $T_{I\times I}$ a cluster tree over $I \times I$ with an admissibility condition, for which $\sigma \times \sigma$ is inadmissible for all $\sigma \subset I$. Then M is also an $\mathcal{H}(T_{I\times I}, k)$-matrix with $k = r$. Moreover, M is in $\mathcal{H}_\ell(r)$.

Remark 2.5.6: The opposite direction of the corollary is generally only true for $r \geq k$. If $M \in \mathcal{H}_\ell(k)$, then M is a semiseparable matrix of rank $\left(\frac{2n}{n_{\min}} - 1\right) r$.

There is a strong relation between tridiagonal matrices and semiseparable matrices, since the inverse of an invertible semiseparable matrix is a tridiagonal matrix as seen, for example, in [96]. This relationship can be used to form efficient eigenvalue algorithms for semiseparable matrices. For instance, there are eigenvalue algorithms for semiseparable matrices that are, for example, based on:

- transformation into a similar tridiagonal matrix [77],
- QR algorithm [102] resp. LR Cholesky algorithm [88] for semiseparable matrices, or
- divide-and-conquer algorithm [27, 78].

The data-sparsity of semiseparable matrices is used in the eigenvalue algorithms which have quadratic complexity for the computation of all eigenvalues. This is relevant here, since we cannot expect to get better results for hierarchical matrices than for semiseparable matrices, due to the inclusion in Corollary 2.5.5.

2.5.3 Hierarchically Semiseparable Matrices

Hierarchically semiseparable matrices combine (as the name suggests) ideas from semiseparable matrices and from hierarchical matrices. Hierarchically semiseparable matrices, HSS matrices for short, have the block structure of a hierarchical matrix with weak admissibility condition. Further, the block rows and block columns span a common low rank space, meaning that hierarchically semiseparable matrices form a subset of the $\mathcal{H}^2$-matrices.

The idea of hierarchically semiseparable matrices was first presented in [26], where it was used to construct a fast and stable solver for the Nyström's discretization of an integral operator of second kind. In recent years there have been more recent publications on equation solvers for HSS matrices [29, 30, 38, 109].

Before we look at the definition of HSS matrices, we should have a look at the following 4×4 HSS matrix [28, (2.2)]:

$$\begin{bmatrix} \begin{bmatrix} D_1 & U_{2;1}B_{21}V_{2;2}^T \\ U_{2;2}B_{22}V_{2;1}^T & D_2 \end{bmatrix} & \begin{bmatrix} U_{2;1}R_{21} \\ U_{2;2}R_{22} \end{bmatrix} B_{11} \begin{bmatrix} W_{23}^T V_{2;3}^T & W_{24}^T V_{2;4}^T \end{bmatrix} \\ \begin{bmatrix} U_{2;3}R_{23} \\ U_{2;4}R_{24} \end{bmatrix} B_{11} \begin{bmatrix} W_{21}^T V_{2;1}^T & W_{22}^T V_{2;2}^T \end{bmatrix} & \begin{bmatrix} D_3 & U_{2;3}B_{23}V_{2;4}^T \\ U_{2;4}B_{24}V_{2;3}^T & D_4 \end{bmatrix} \end{bmatrix}.$$

The following definition uses the structural identity between HSS and $\mathcal{H}_\ell$-matrices. This means that this definition is different from the definitions in [29] or [109], but defines the same set of matrices.

Definition 2.5.7: (HSS matrices)
Let $M \in \mathcal{H}_\ell(k)$. To shorten the notation we use $M_{j;r,s}$ for the submatrix of size $2^{\ell-j}n_{\min} \times 2^{\ell-j}n_{\min}$. There are $2^j \times 2^j$ such submatrices indexed by (r,s) from $(1,1)$ to $(2^j \times 2^j)$. The low rank factorization of $M_{j;r,s}$ is

$$M_{j;r,s} = A_{j;r,s}B_{j;r,s}^T.$$

We will call M a hierarchically semiseparable matrix of (HSS) rank k, $M \in \mathrm{HSS}(k)$ for short, if M fulfills the following conditions:

- $|r-s| = 1 \Rightarrow M|_{(r-1)2^{\ell-j}n_{\min}+1:r2^{\ell-j}n_{\min},(s-1)2^{\ell-j}n_{\min}+1:s2^{\ell-j}n_{\min}} = A_{j;r,s}B_{j;r,s}^T$
- $\forall i \in \{1,\ldots,2^\ell\} : \exists U_{\ell;i} \in \mathbb{R}^{n_{\min}\times k}$ and $\exists U_{j;r} \in \mathbb{R}^{2^{\ell-j}n_{\min}\times k}$, $j = 1,\ldots,\ell-1$, $r = 1,\ldots,2^j$ with $\operatorname{range}(U_{j;r}) \subset \operatorname{span}\left(\begin{bmatrix} U_{j+1;2r-1} \\ 0 \end{bmatrix}\right) \oplus \operatorname{span}\left(\begin{bmatrix} 0 \\ U_{j+1;2r} \end{bmatrix}\right)$ and $\operatorname{range}(A_{j;r,s}) \subset \operatorname{span}(U_{j;r})$;
- $\forall i \in \{1,\ldots,2^\ell\} : \exists V_{\ell;i} \in \mathbb{R}^{n_{\min}\times k}$ and $\exists V_{j;s} \in \mathbb{R}^{2^{\ell-j}n_{\min}\times k}$, $j = 1,\ldots,\ell-1$, $s = 1,\ldots,2^j$ with $\operatorname{range}(V_{j;s}) \subset \operatorname{span}\left(\begin{bmatrix} V_{j+1;2s-1} \\ 0 \end{bmatrix}\right) \oplus \operatorname{span}\left(\begin{bmatrix} 0 \\ V_{j+1;2s} \end{bmatrix}\right)$ and $\operatorname{range}(B_{j;r,s}) \subset \operatorname{span}(V_{j;s})$.

In the example above one would define

$$U_{1;2} := \begin{bmatrix} U_{2;3}R_{23} \\ U_{2;4}R_{24} \end{bmatrix}.$$

Corollary 2.5.8: If $M \in \mathbb{R}^{n\times n}$ is a semiseparable matrix of semiseparability rank k, then $M \in \mathrm{HSS}(k)$.

Lemma 2.5.9:

(a) Let $M \in \mathrm{HSS}_\ell(k)$ be an HSS matrix of HSS rank k with a hierarchical structure depth of ℓ. Then $M \in \mathcal{H}_\ell(k)$.

(b) Further, if $N \in \mathcal{H}_\ell(k)$ is an $\mathcal{H}_\ell$-matrix of block-wise rank k and depth ℓ, then $N \in \mathrm{HSS}_\ell(k\ell)$.

Proof. Part (a): trivial. Part (b): There are parts from ℓ blocks and the inadmissible diagonal block in each block row. Each block is of rank k, meaning that the block row except the diagonal block is of rank at most $k\ell$.

■

The HSS matrices are somehow of one-dimensional structure since the discretization of one-dimensional problems primarily leads to HSS matrices. This makes the structure of HSS matrices not very rich. They are mentioned here only for completeness.

2.6 Review of Existing Eigenvalue Algorithms

The aim of this section is to give a short review on other approaches for the computation of eigenvalues of hierarchical matrices as Chapter 6 is on vector and subspace iterations. Several authors have used power and inverse iterations to compute eigenvalues of $\mathcal{H}$-matrices or norms. These existing algorithms will be reviewed there.

2.6.1 Projection Method

In [60] Hackbusch and Kreß present a projection method that computes a matrix $\tilde{M}_p$. The eigenvalues of M, which are contained in the interval (a, b), are (almost) retrieved in $\tilde{M}_p$. The other eigenvalues are projected to (almost) zero. The effect of this is that a subspace iteration on $\tilde{M}_p$ converges really fast. Besides this, one can use subspace iteration for the computation of inner eigenvalues.

The best projection is

$$\chi : \mathbb{R} \to \mathbb{R} : \chi(\lambda) = \begin{cases} \lambda, & \lambda \in (a, b), \\ 0, & \text{else.} \end{cases}$$

As this projection requires the computation of the invariant subspace of the eigenvalues in (a, b), it is too expensive. In [60] the following approximation is used

$$\tilde{\chi}(\lambda) = \frac{\lambda}{1 + T(\lambda)^{2^p}} \text{ with } T(\lambda) = \frac{2\lambda - (b + a)}{b - a}.$$

The projected matrix $\tilde{M}_p$ is then computed using

$$\tilde{M}_p := \tilde{\chi}(M) = (I + T(M)^{2^p})^{-1} M.$$

This computation is relatively cheap in $\mathcal{H}$-arithmetic. The effect of the projection is shown in Figure 2.12. The green marked eigenvalues of the projected matrix are problematic, since they lie in (a, b). The projection does not effect the eigenvectors, so we can compute an invariant subspace X of $\tilde{M}_p$ and use the generalized matrix Rayleigh quotient $X^T M X$ for the computation of the sought eigenvalues.

For large p, say $p = 3$ or 4, the inversion of $(I + T(M)^{2^p})$ is badly conditioned. This leads to large errors in the computed eigenvalues. Hackbusch and Kreß use a preprocessing step to cure this problem.

The author implemented this method without preprocessing. The eigenvalues are computed by subspace iteration. The subspace dimension is chosen to be twice the number of eigenvalues in (a, b). The iteration is stopped if the residual of the eigenvalue problem for M falls below ϵ:

$$\|R\|_2 = \left\| MX - XX^T MX \right\|_2 < \epsilon,$$

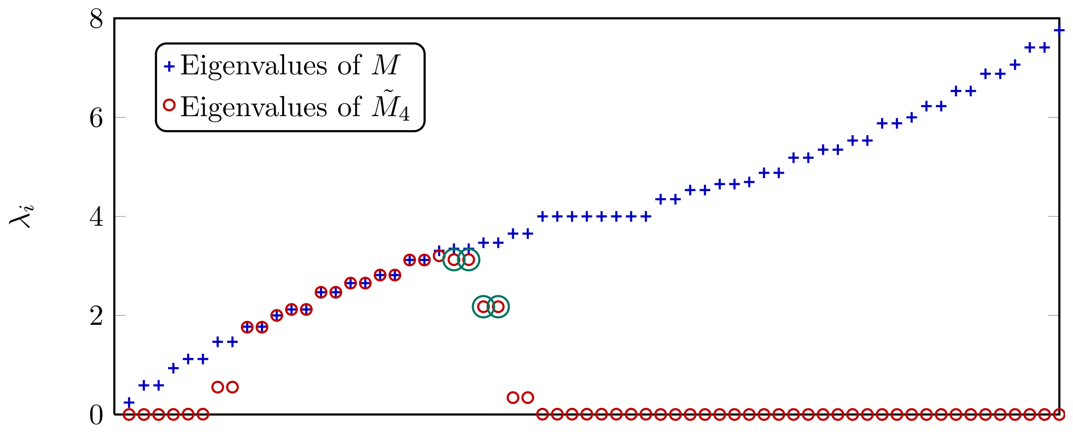


Figure 2.12: Eigenvalues of matrix M = FEM8 before and after projection ($p = 4$, $a = 1.5$, $b = 3.5$).

with X being an orthogonal basis for the subspace in the current iterate. In Chapter 7 this implementation will be compared to the LDL^T slicing algorithm presented in Chapter 5.

2.6.2 Divide-and-Conquer for $\mathcal{H}_\ell(1)$-Matrices

In [47] J. Gördes uses a divide-and-conquer method for the computation of the eigenvalues of an $\mathcal{H}_\ell(1)$-matrix. Like the solver described in [19], this method leads to an algorithm of almost quadratic complexity.

The main idea of the divide-and-conquer method for symmetric tridiagonal matrices [32] is that every tridiagonal matrix T can be writen as

$$T = \begin{bmatrix} T_1 & 0 \\ 0 & T_2 \end{bmatrix} + \alpha b b^T,$$

with $T_1 \in \mathbb{R}^{n_1 \times n_1}, T_2 \in \mathbb{R}^{n_2 \times n_2}$ tridiagonal, and

$$b_i = \begin{cases} 1, & i = n_1 \text{ or } i = n_1 + 1, \\ 0, & \text{else.} \end{cases}$$

The $\mathcal{H}_\ell(1)$-matrices have a similar block recursive structure of two rank-1 perturbations. Here two rank-1 perturbations are necessary, as otherwise the local rank of the diagonal submatrices would be increased by 1. Further, the $\mathcal{H}_\ell$-matrix-structure does not permit an easy computation of the eigenvectors, as it is done in [27] for semiseparable matrices. The computation of the eigenvectors is expensive, so only with the solver in [19] the costs are quadratic.

The main drawback is the limitation of $\mathcal{H}_\ell(1)$-matrices. A generalization to $\mathcal{H}_\ell(k)$-matrices seems to be expensive. There are currently no ideas how to generalize this

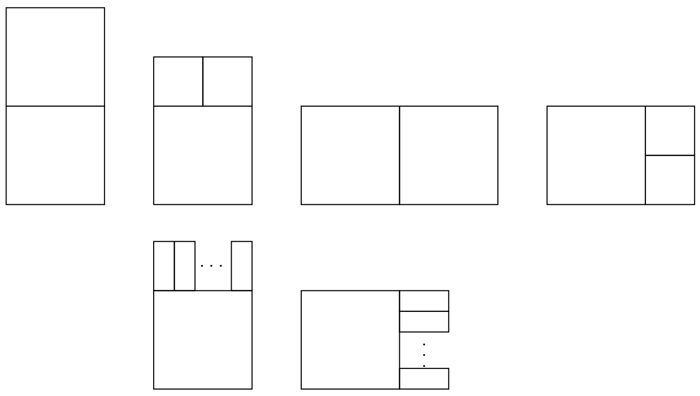


Figure 2.13: Block structure condition [34].

method to $\mathcal{H}$-matrices based on the standard admissibility condition, since the rank of $M_{(I\setminus\tau)\times\tau}$ is large for $M \in \mathcal{H}(T_{I\times I}, k)$.

2.6.3 Transforming Hierarchical into Semiseparable Matrices

In [34] an algorithm is described that transforms a hierarchical semiseparable matrix or an $\mathcal{H}^2$-matrix with a special condition on the block structure into a semiseparable matrix in Givens-vector representation, requiring $\mathcal{O}(k^3 n \log_2 n)$ flops. The eigenvalues of semiseparable matrices in Givens-vector representation can be computed by a QR algorithm [39] or an LR Cholesky algorithm [88]. Both eigenvalue computations require $\mathcal{O}(n^2)$ flops.

The transformation does not work for general $\mathcal{H}$-matrices. Furthermore, the special condition on the block structure is only fulfilled by a small subset of $\mathcal{H}^2$-matrices. This condition demands that the $\mathcal{H}^2$-matrix has a block structure, which becomes coarser to the lower left corner and, due to symmetry, to the upper right corner. In the lower triangle, neighboring blocks have to resemble the pictures in Figure 2.13. In the upper triangle we assume a structure symmetric to the lower triangle.

One can show that the discretization of one-dimensional integral equations can lead to a hierarchical block structure that will fulfill this condition. For higher dimensional problems it seems to be much more difficult to get a block structure as shown in Figure 2.13.

The $\mathcal{H}_\ell$-matrices have this block structure, such that they can be transformed into a semiseparable matrix. This transformation is explained in Subsection 4.3.5, but as we see there, the transformation is not necessary, since we can perform the LR Cholesky algorithm directly on the $\mathcal{H}_\ell$-matrices.

Summarizing this section, one can say that there are two eigenvalue algorithms for $\mathcal{H}_\ell$-matrices and other simple structured $\mathcal{H}$-matrices and one eigenvalue algorithm for general $\mathcal{H}$-matrices computing some inner eigenvalues. In the following chapters new

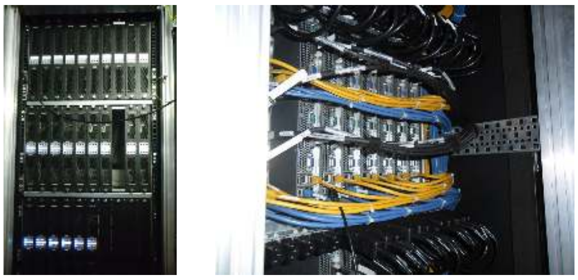


Figure 2.14: Photos of Linux-cluster Otto.

algorithms for general $\mathcal{H}$- and $\mathcal{H}_\ell$-matrices are investigated. This chapter is closed by a short description on the computing facilities we use for the numerical computations.

2.7 Compute Cluster Otto

The numerical computations, which are presented at the end of the chapters and in Chapter 7, were all done on the *Max Planck Institute for Dynamics of Complex Technical Systems'* compute cluster Otto. One node of Otto has two Intel®Xeon®Westmere X5650 with 2.66GHz and 48GB DDR3 RAM. Each X5650 has 6 cores. There are also nodes with AMD processors, but they are not used for the numerical computations within this thesis. Except for the computations for parallelization of the slicing algorithm, see Chapter 5, all the computations were performed only on a single core of a single node. We take care that no other computation runs simultaneously on the same node. In summary, most of the numerical computations are done in a setting with performance characteristics comparable to desktop computers.

Nevertheless, the parallel computations in Chapter 5 use the cluster to its full capability. In one example we use up to 32 nodes with totally 384 cores. Each core investigates its own part of the spectrum. Here we take care that the 4GB RAM provided per core are sufficient for the computations. We do not use a parallel version of the $\mathcal{H}$Lib [65].

In this chapter the essential basics for the remainder of this thesis are discussed. The next chapter is on the QR decomposition of hierarchical matrices. This is again a chapter that does not deal directly with the eigenvalue problem for hierarchical matrices, but instead we will use the $\mathcal{H}$-QR decomposition in Chapter 4 to construct a QR algorithm for $\mathcal{H}$-matrices.

CHAPTER

THREE

QR DECOMPOSITION OF HIERARCHICAL MATRICES

Contents

3.1 Introduction

The QR decomposition, see Equation (2.4), is used in the explicit QR algorithm, which will be applied to $\mathcal{H}$-matrices in the next chapter. So we require a QR decomposition for $\mathcal{H}$-matrices. Further QR decompositions are used in a variety of matrix algebra problems (e.g., least squares problems and linear systems of equations). We should have available as many arithmetic operations as possible for hierarchical matrices and therefore it is also useful to have a QR decomposition for hierarchical matrices.

We will call a QR decomposition

$$QR = M \in \mathcal{H}(T,k),$$

with $Q, R \in \mathcal{H}(T,k')$ an *$\mathcal{H}$-QR decomposition*. The matrix R has to be an upper triangular $\mathcal{H}$-matrix, see Definition 2.3.11. Since many arithmetic operations for $\mathcal{H}$-matrices have linear-polylogarithmic complexity, the $\mathcal{H}$-QR decomposition should also have such a complexity.

Besides having a complexity that is to be expected for formatted arithmetic operations, the $\mathcal{H}$-QR decomposition should be a good orthogonal decomposition, too. Firstly that means

$$r_{\text{QR}} := \|QR - M\|_2 \tag{3.1}$$

should be small, and secondly, that the matrix Q should be orthogonal. We will call Q a *nearly orthogonal* matrix if

$$r_{\text{orth}}(Q) := \left\|Q^T Q - I\right\|_2 \tag{3.2}$$

is small.

So far, two $\mathcal{H}$-QR decompositions have been suggested in the literature, see [6, 73]. Both have deficiencies in one of the above requirements as we will see later in Section 3.4. Therefore, we will investigate an alternative approach that does not yet overcome all difficulties, but offers some advantages over the existing methods to compute the $\mathcal{H}$-QR decomposition.

The outline of this chapter is as follows. In the next section we will briefly review the two known methods to compute an $\mathcal{H}$-QR decomposition. The following section focuses on the presentation of a new $\mathcal{H}$-QR decomposition including some complexity estimates and a comparison with sparse QR decompositions. In Section 3.4 we compare the three methods using some numerical tests. Some concluding remarks will be given in the end.

This chapter is based in large parts on [11].

3.2 Review of Known QR Decompositions for $\mathcal{H}$-Matrices

In this section we will review the two existing $\mathcal{H}$-QR decompositions. After the presentation of the new $\mathcal{H}$-QR decomposition we will compare the three using three numerical examples.

3.2.1 Lintner's $\mathcal{H}$-QR Decomposition

Let $M = QR$ be the QR decomposition of M. Obviously

$$M^T M = R^T Q^T Q R \overset{Q^TQ=I}{=} R^T R \tag{3.3}$$

Algorithm 3.1: Lintner's $\mathcal{H}$-QR Decomposition.

```
    Input: $M$
    Output: $Q, R$, with $M = QR$
1   $B := M^T *_{\mathcal{H}} M$;
2   $R^T := \mathcal{H}$-Cholesky factorization($B$);
3   Solve $M = QR$;                    /* $\mathcal{H}$Lib function SolveLeftCholesky */
```

Algorithm 3.2: Polar Decomposition [63].

```
    Input: $M$
    Output: $Q, P$, with $M = QP$
1   $Q_0 := M$;
2   $Q_{i+1} := \frac{1}{2}\left(\gamma_i Q_i + \frac{1}{\gamma_i} Q_i^{-T}\right)$;  /* $\gamma_i \neq 1$ improves convergence, see [63]  */
3   $Q := Q_\infty$; $P := Q^T M$;
```

holds. In his dissertation thesis [73] and later in [74], Lintner describes an $\mathcal{H}$-QR decomposition, which first computes R by the Cholesky factorization of $M^T M$ and then Q by solving an upper triangular system of equations. This leads to Algorithm 3.1, the first algorithm for computing an $\mathcal{H}$-QR decomposition.

This algorithm consists only of well known hierarchical operations of linear-polylogarithmic complexity. The solution of a linear upper triangular system is part of the Cholesky decomposition. This $\mathcal{H}$-QR decomposition can be implemented easily using the $\mathcal{H}$Lib [65], since the three required functions are included.

As the matrix R is computed without any care to the orthogonality of Q, we can not expect a nearly orthogonal matrix Q. Indeed in many examples, see Section 3.4, the computed Q is far from being orthogonal. Lintner suggests to compute the $\mathcal{H}$-QR decomposition of Q yielding

$$M = QR = Q'R'R = Q'(R'R)$$

and to use Q' times $R'R$ as QR decomposition. We hope that $r_{\text{orth}}(Q') < r_{\text{orth}}(Q)$. If Q' is still not orthogonal enough, we may repeat this reorthogonalization process.

The condition number of $M^T M$ is

$$\kappa(M^T M) \approx \kappa(M)^2.$$

For badly conditioned problems, squaring the condition number increases the error-sensitivity dramatically and causes the low orthogonality.

Lintner uses the polar decomposition to reduce the condition number to the square root of $\kappa(M)$, see Algorithm 3.2. The polar decomposition is computed by an iterative method similar to the sign-function iteration, see [63]. One can show that the polar decomposition of an $\mathcal{H}$-matrix can be computed in almost linear complexity. In each iteration step an $\mathcal{H}$-inversion is needed, this makes the polar decomposition expensive. Lintner suggests the following steps: first determine the polar decomposition $M = QM'$, then compute the Cholesky factorization $M' = R^T R$, and finally use Algorithm 3.1 to decompose $R^T = Q'R'$. If we insert these equations, we get

$$M = QM' = QR^T R = QQ'R'R \approx \left(Q *_{\mathcal{H}} Q'\right)\left(R' *_{\mathcal{H}} R\right).$$

Multiplying Q by Q' gives the orthogonal and R' by R the upper triangular factor of the $\mathcal{H}$-QR decomposition of M. Lintner shows that the condition number of the last step is

$$\kappa\left(R^T\right) = \sqrt{\kappa\left(M\right)}.$$

So far the biggest disadvantage, the squaring of the condition number, is more than made up. In the next subsection we will see a second possibility to avoid squaring the condition number. First, a remark on the solution of the linear least squares problem:

Remark 3.2.1: If we are interested in the solution of the linear least squares problem, then this method is not the best one, since it will compute the solution using the normal equations and this is dangerous for ill conditioned matrices M, see [64, p. 386ff].

3.2.2 Bebendorf's $\mathcal{H}$-QR Decomposition

Algorithm 3.3 is described in [6, p. 87ff]. This algorithm computes a series of orthogonal transformations that triangularize M. On the first recursion level the matrix is transformed to a block upper triangular form. The two resulting diagonal blocks are triangularized by recursion. In line 10 of Algorithm 3.3 the orthogonal transformation on the current level is described by the product

$$\begin{bmatrix} L_1^{-1} & 0 \\ 0 & L_2^{-1} \end{bmatrix} \begin{bmatrix} I & X^T \\ -X & I \end{bmatrix} = \begin{bmatrix} L_1^{-1} & L_1^{-1}X^T \\ -L_2^{-1}X & L_2^{-1} \end{bmatrix}.$$

This is a kind of block Givens rotation, because

$$\det \begin{bmatrix} L_1^{-1} & L_1^{-1}X^T \\ -L_2^{-1}X & L_2^{-1} \end{bmatrix} = \det L_1^{-1}\ \det\left(L_2^{-1}\left(I + XX^T\right)\right) = 1.$$

On each recursion level the matrix M_{11} has to be inverted. This probably makes the algorithm expensive. Further, not all matrices have an invertible first diagonal block, i.e.,

$$\begin{bmatrix} 0 & I \\ I & 0 \end{bmatrix}$$

has a QR decomposition, but 0 is not invertible.

Algorithm 3.3: Bebendorf's $\mathcal{H}$-QR Decomposition.

```
   Input: $M = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix}$
   Output: $Q, R$, with $M = QR$
1  Function $[Q, R] = \mathcal{H}$-QR $(M)$ begin
2      if $M \notin \mathcal{L}(T)$ then
3          $X := M_{21} *_{\mathcal{H}} (M_{11})^{-1}_{\mathcal{H}}$;
4          $L_1 L_1^T := \mathcal{H}$-Cholesky factorization$(I + X^T X)$;
5          $L_2 L_2^T := \mathcal{H}$-Cholesky factorization$(I + X X^T)$;
6          $R := \begin{bmatrix} L_1^T M_{11} & L_1^{-1}\left(M_{12} + X^T M_{22}\right) \\ 0 & L_2^{-1}\left(M_{22} - X M_{12}\right) \end{bmatrix} = \begin{bmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{bmatrix}$;
7          $[Q_1, R_{11}] := \mathcal{H}$-QR $(R_{11})$;
8          $R_{12} := Q_1^T R_{12}$;
9          $[Q_2, R_{22}] := \mathcal{H}$-QR $(R_{22})$;
10         $Q := \begin{bmatrix} I & X^T \\ -X & I \end{bmatrix}^T *_{\mathcal{H}} \begin{bmatrix} L_1^{-1} & 0 \\ 0 & L_2^{-1} \end{bmatrix}^T *_{\mathcal{H}} \begin{bmatrix} Q_1 & 0 \\ 0 & Q_2 \end{bmatrix}$;
11     end
12     else $[Q, R] :=$ QR$(A)$ ;                    /* standard QR decomposition */
13 end
```

Bebendorf shows that the complexity of Algorithm 3.3 is determined by the complexity of the $\mathcal{H}$-matrix multiplication, so this is also an algorithm of linear-polylogarithmic complexity. He further shows that increasing the $\mathcal{H}$-arithmetic precision from level to level as ϵ/l ensures $r_{\text{orth}}(Q) \sim \epsilon \log\left(\text{depth}(T_{I\times I})\right)$.

Remark 3.2.2: If we assume that the block A_{21} has the structure

$$A_{21} = \begin{bmatrix} 0 & \cdots & 0 & * \\ 0 & \cdots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & 0 \end{bmatrix},$$

as may be the case in QR iterations of a Hessenberg-like structured $\mathcal{H}$-matrix, then inverting the last diagonal block of A_{11} is enough to zero A_{21}. The matrix Q then has the structure

$$Q = \begin{bmatrix} I &&&&&&& \\ & \ddots &&&&&& \\ && I &&&&& \\ &&& L_1^{-1} & L_1^{-1}X^T &&& \\ &&& -L_2^{-1}X & L_2^{-1} &&& \\ &&&&& I && \\ &&&&&& \ddots & \\ &&&&&&& I \end{bmatrix}.$$

This leads to a considerable reduction of cost. For this Q, it is easy to see that we use a block generalization of Givens rotations.

3.3 A new Method for Computing the $\mathcal{H}$-QR Decomposition

In this section we present a new $\mathcal{H}$-QR decomposition. The main idea is to use the standard QR decomposition for dense matrices as often as possible. This new algorithm works recursively dividing the matrix into block-columns.

First, we will investigate what to do on the lowest level of the hierarchical structure. Later, we will describe the recursive computation for non-leaf block-columns. On these higher levels we will use a block modified Gram-Schmidt orthogonalization (MGS). In particular we demonstrate that MGS can be used in the hierarchical matrix format and leads to an orthogonalization process of almost linear complexity. Hence, we can reduce the usual $\mathcal{O}(n^3)$ complexity of MGS when applied to dense matrices significantly. For this block-orthogonalization we use only $\mathcal{H}$-matrix-matrix multiplications and additions. In contrast to the two other $\mathcal{H}$-QR decompositions, we do not use the expensive $\mathcal{H}$-inversion.

3.3.1 Leaf Block-Column

Let M be an $\mathcal{H}$-matrix and $T_{I\times I}$ the corresponding $\mathcal{H}_\times$-tree based on $T_I \times T_I$. We will call a block-column $M_{I\times s}$ a leaf block-column if s is a leaf of T_I, $s \in \mathcal{L}(T_I)$. In this subsection we will compute the QR decomposition of such a leaf block-column.

Our leaf block-column $M_{I\times s}$ consists of blocks M_i. These blocks are elements of the block-hierarchical tree $\mathcal{H}_\times$ of our $\mathcal{H}$-matrix. In this subsection we will assume that the block $r_i \times s$ corresponding to M_i includes a leaf r_i of our $\mathcal{H}$-tree. The other cases will be treated in the next subsection.

The block $r_i \times s$ is an admissible or a non-admissible block. If the block is non-admissible, we will treat this block like an admissible one by substituting M_i by $I\left(M_i^T\right)^T$ or $M_i I$.

So all blocks of M have the structure $A_iB_i^T$,

$$M = \begin{bmatrix} M_1 \\ M_2 \\ \vdots \\ M_p \end{bmatrix} = \begin{bmatrix} A_1B_1^T \\ A_2B_2^T \\ \vdots \\ A_pB_p^T \end{bmatrix}.$$

We now write M itself as a product of two matrices:

$$M = AB^T = \begin{bmatrix} A_1 & 0 & \cdots & 0 \\ 0 & A_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & A_p \end{bmatrix} \begin{bmatrix} B_1 & B_2 & \dots & B_p \end{bmatrix}^T.$$

We notice that B is a dense matrix.

We will compute the QR decomposition of this block-column in two steps. First we transform A to an orthogonal matrix and after that we compute the QR decomposition of the resulting B'.

The matrix A is orthogonalized block-by-block. Every A_i has to be factorized into $Q_{A_i}R_i$ using the standard QR decomposition, since A_i is dense. The matrices R_i will be multiplied from the right hand side to B_i, $B_i' = B_iR_i^T$.

The second step is as simple as the first one. The matrix B'^T is dense, so we use the standard QR decomposition for the decomposition of $B'^T = Q_B^TR_B$. We get an orthogonal matrix Q_AQ_B and an upper triangular matrix R_B:

$$M = AB^T = \begin{bmatrix} Q_{A_1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & Q_{A_p} \end{bmatrix} \begin{bmatrix} R_1B_1^T \\ \vdots \\ R_pB_p^T \end{bmatrix} = \begin{bmatrix} Q_{A_1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & Q_{A_p} \end{bmatrix} Q_BR_B.$$

The resulting matrix Q_B^T is subdivided like B^T, denoting the blocks $Q_{B_i}^T$. We combine the matrices $Q_{A_i}Q_{B_i}^T$ again and get a block-column matrix Q with the same structure as M,

$$M = Q_AQ_BR_B = \underbrace{\begin{bmatrix} Q_{A_1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & Q_{A_p} \end{bmatrix} \begin{bmatrix} Q_{B_1}^T \\ \vdots \\ Q_{B_p}^T \end{bmatrix}}_{=:Q} R_B. \qquad (3.4)$$

We have

$$\begin{aligned} Q^TQ &= Q_B^T\mathrm{diag}\left\{Q_{A_1},\dots,Q_{A_p}\right\}^T \mathrm{diag}\left\{Q_{A_1},\dots,Q_{A_p}\right\} Q_B \\ &= Q_B^T \underbrace{\mathrm{diag}\left\{Q_{A_1}^TQ_{A_1},\dots,Q_{A_p}^TQ_{A_p}\right\}}_{=\mathrm{diag}\{I,\dots,I\}} Q_B = Q_B^TQ_B = I \end{aligned}$$

Algorithm 3.4: $\mathcal{H}$-QR Decomposition of a Block-Column.

```
Input: $M_{I\times s} = \left[A_i B_i^T\right]_{i=1}^p$
Output: $Q, R$, with $M_{I\times s} = QR$
for $i = 1, \dots, p$ do
    if $M_i$ is admissible then
        $[Q_{A_i}, R_i] :=$ QR($A_i$);
        $A_i := Q_{A_i}$ and $B_i^T := R_i B_i^T$;
    else
        $M_i \in \mathbb{R}^{c_i \times d_i}$;
        if $c_i \leq d_i$ then $B_i^T = M_i$ ($A_i := I$);   /* $A_i$ is already orthogonal */
        else
            $[Q_{A_i}, R_i] :=$ QR($M_i$);                  /* e.g. LAPACK dfeqrf [2] */
            $A_i := Q_{A_i}$ and $B_i^T := R_i$;
        end
    end
end
Assemble $B^T = \begin{bmatrix} B_1 & B_2 & \dots & B_p \end{bmatrix}^T$;
$[Q_B, R_B] :=$ QR($B^T$);
Partition $Q_B$ into blocks as in (3.4), $B_i^T := Q_{B_i}^T$;
for $i = 1, \dots, p$ do
    if $M_i$ is admissible then $A_i B_i^T$ is a block of $Q$;
    else if $A_i = I$ then $B_i^T$ is a dense block of $Q$;
    else Compute $A_i B_i^T$ to get a dense block of $Q$;
end
```

so that Q is orthogonal.

These steps are described in Algorithm 3.4. We only use matrix-matrix multiplications and QR decompositions for standard matrices, so the accuracy of the computation can be regarded as perfect when compared with the approximation error of the $\mathcal{H}$-arithmetic.

3.3.2 Non-Leaf Block Column

In the last subsection we have factorized a leaf block-column $M_{I\times s}$, $s \in \mathcal{L}(T_I)$. In this subsection we will use this to recursively compute the $\mathcal{H}$-QR decomposition of a hierarchical matrix. In general an $\mathcal{H}$-matrix $M \in \mathbb{R}^{I\times I}$ is a non-leaf block-column.

Let $M_{I\times t}$ be a non-leaf block column of M, that means $S(t) \neq \emptyset$. We have chosen the numbering of the indices, so that we can order the sons $s_i \in S(t)$:

$$s_1 < s_2 < s_3 < \cdots ,$$

see Definition 2.3.11 and the paragraph before.

We get the QR decomposition of the first block-column $M_{I\times s_1}$ by using this recursion. The QR decomposition of the second block-column starts with the orthogonalization w.r.t. the first block-column by using a block modified Gram-Schmidt orthogonalization. We compute

$$\begin{aligned} R_{s_1\times s_2} &= Q_{I\times s_1}^T *_{\mathcal{H}} M_{I\times s_2} \qquad\qquad \text{and} \\ M'_{I\times s_2} &:= M_{I\times s_2} -_{\mathcal{H}} Q_{I\times s_1} *_{\mathcal{H}} R_{s_1\times s_2}. \end{aligned}$$

Now we apply again the recursion to compute the QR decomposition of $M'_{I\times s_2}$. If t has more than two sons, the other block-columns can be treated analogously. Algorithm 3.5 describes these steps in algorithmic form.

We have chosen the $\mathcal{H}_\times$-tree in order to find large admissible blocks. Let $r_a \times t$ be an admissible block and $C \in \mathbb{R}^{s_i\times t} = AB^T$ the corresponding **R**k-matrix. We must split C into two submatrices before we continue with the steps above. We will do something similar to Algorithm 3.4 to get a dense, easily partitionable matrix: computing the standard QR decomposition of $A = Q_A R_A$, multiplying $R_A B^T$ and storing Q_A. Using Q_A we factorize our block-column as

$$M_{I\times t} = \begin{bmatrix} I & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & Q_A & 0 \\ 0 & \cdots & 0 & I \end{bmatrix} \begin{bmatrix} M_{r_1\times s_1} & M_{r_1\times s_2} \\ M_{r_2\times s_1} & M_{r_2\times s_2} \\ \vdots & \vdots \\ \multicolumn{2}{c}{R_A B^T} \\ \vdots & \vdots \end{bmatrix}.$$

The matrix $R_A B^T \in \mathbb{R}^{k\times|t|}$ is a small rectangular dense matrix. Splitting a dense matrix into two block-columns in a columnwise organized storage simply means setting a second pointer to the first element of the second matrix. Note: on the next levels we have to split this dense matrix again.

Apart from the adaptively chosen ranks of the admissible blocks, the matrix Q has the same structure as M. The matrix R is also an $\mathcal{H}$-matrix. In the next section we will investigate the complexity of this algorithm.

3.3.3 Complexity

The following theorem analyzes the complexity of the $\mathcal{H}$-QR decomposition.

Theorem 3.3.1: Let M be an $\mathcal{H}$-matrix with the minimum block size $n_{\min}$. Then Algorithm 3.5 has a complexity of $\mathcal{O}(k_{\max}^2 n (\log_2 n)^2)$ with

$$k_{\max} = \max_{r\times s\in\mathcal{L}^+(T_{I\times I})} \max\left\{\operatorname{rank}(M_{r\times s}), \operatorname{rank}(Q_{r\times s}), \operatorname{rank}(R_{r\times s})\right\}.$$

Algorithm 3.5: $\mathcal{H}$-QR Decomposition of an $\mathcal{H}$-Matrix.

```
   Input: $M_{I\times t}$, $\{s_1,\dots,s_q\} \in S(t)$
   Output: $Q \in \mathbb{R}^{I\times t}, R \in \mathbb{R}^{t\times t}$, with $M = QR$
 1 if $t \in \mathcal{L}(T)$ then
 2     Compute the QR decomposition using Algorithm 3.4;
 3 else
 4     for $k = 1,\dots,p$ do
 5         if $r_k \times t$ is admissible then $[Q_{r_k\times t}, R_A] :=$ QR($A_{r_k\times t}$); $B^T_{r_k\times t} := R_A B^T_{r_k\times t}$;
 6         Split all dense matrices that overlap more than one block-column,
           according to the block-partitioning given by the $\mathcal{H}$-tree;
           /* see Figure 3.1                                                    */
 7     end
 8     for $j = 1,\dots,q$ do
 9         for $i = 1,\dots,j-1$ do          /* modified Gram-Schmidt orthog.  */
10             $R_{s_i\times s_j} := Q^T_{I\times s_i} M_{I\times s_j}$;
11             $M_{I\times s_j} := M_{I\times s_j} - Q_{I\times s_i} R_{s_i\times s_j}$;
12         end
13         Compute the QR decomposition of $M_{I\times s_j}$ recursively;
14     end
15     for $k = 1,\dots,p$ do
16         if $r_k \times t$ is admissible then use the stored $Q_{r_k\times t}$ and the computed
           rectangular matrix $B'$ to form the Rk-matrix $QB'^T$;
17         Recombine overlapping matrices;                    /* see Figure 3.2 */
18     end
19 end
```

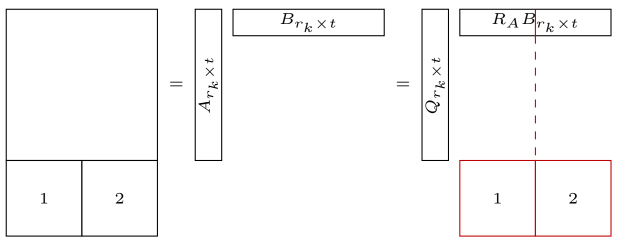


Figure 3.1: Explanation of Line 5 and 6 of Algorithm 3.5.

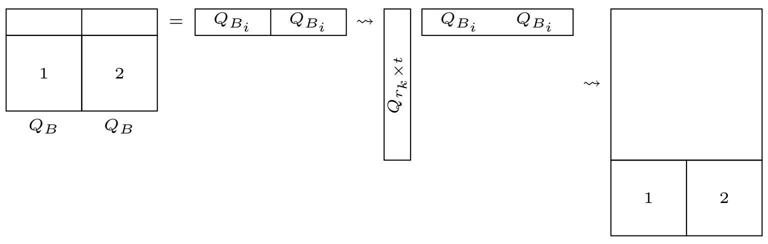


Figure 3.2: Explanation of Line 16 and 17 of Algorithm 3.5.

Proof. For all $r \times s \in \mathcal{L}^+$ we have $M_{r\times s} = A_{r\times s}B_{r\times s}^T$. Each matrix $A_{r\times s}$ is decomposed using the standard QR decomposition. The matrix $A_{r\times s}$ has the dimension $m_r \times k$, with k the rank of the block $r \times s$. The standard QR decomposition of a matrix $F \in \mathbb{R}^{e\times f}$ needs $\mathcal{O}(f^2 e)$ flops, so the QR decomposition of $A_{r\times s}$ needs $\mathcal{O}(k^2 m_r)$ flops. The matrix $A_{r\times s}$ needs $N_{A_{r\times s},st} = \mathcal{O}(m_r k)$ storage. If we sum over all admissible leaves $r \times s \in \mathcal{L}^+$, we get

$$\sum_{r\times s\in\mathcal{L}^+} N_{\mathrm{QR}(A_{r\times s})} \in \mathcal{O}\left(k_{\max} N_{\mathcal{H},st}\right).$$

Analogously, the number of flops required for line 9 in Algorithm 3.4 is in $\mathcal{O}\left(n_{\min} N_{\mathcal{H},st}\right)$.

For each block-column $M_{I\times s}$ of the lowest level we compute another QR decomposition to treat the remaining factor of compounded B_i. The vertex s should have at most $n_{\min}$ indices, otherwise the non-admissible diagonal block of M would be divided once more. It follows that such a block-column has at most $n_{\min}$ columns, $|s| \leq n_{\min}$. Further we need the number of rows ρ of B^T. The block-column is composed of matrix-blocks M_i (and parts of such blocks) of the form $r_i \times s_i$, with $r_i \times s_i \in \mathcal{L}(T_{I\times I})$ and $s \in S^*(s_i)$. The number of rows ρ is the sum of the ranks of $M_{r_i\times s_i}$ and the size of the non-admissible blocks,

$$\rho = \sum_{\substack{r_i\times s_i\in\mathcal{L}^+(T_{I\times I})\\ s\in S^*(s_i)}} \operatorname{rank}\left(M_{r_i\times s_i}\right) + \sum_{\substack{r_i\times s_i\in L^-(T_{I\times I})\\ s\in S^*(s_i)}} \min\left\{|r_i|, |s_i|\right\}.$$

In the non-admissible leaves we use the test in line 7 in Algorithm 3.4 to ensure that the number of rows of the corresponding B_i^T is $\min\{|r_i|, |s_i|\} \leq n_{\min}$. The condition $s \in S^*(s_i)$ means that s_i is a father or an ancestor of s. There is only one s_i on each level fulfilling this condition. The Definition 2.3.7 of the sparsity constant C_{sp} gives

$$\sum_{\substack{r_i\times s_i\in\mathcal{L}^+(T_{I\times I})\\ s\in S^*(s_i)}} \operatorname{rank}\left(M_{r_i\times s_i}\right) \leq C_{sp}\operatorname{depth}(T_{I\times I})k_{\max}.$$

So we can bound ρ by

$$\rho \leq C_{sp}k_{\max}\mathcal{O}(\log_2 n) + C_{sp}n_{\min}.$$

This bound on the number of rows is now used to bound the costs of this QR decompositions by

$$N_{\mathrm{QR}(B,\text{one column})} \leq C_{sp}\left(k_{\max}\mathcal{O}(\log_2 n) + n_{\min}\right) n_{\min}|s|.$$

Summing over all columns gives

$$N_{\mathrm{QR}(B,\text{all})} \leq C_{sp}\left(k_{\max}\mathcal{O}(\log_2 n) + n_{\min}\right) n_{\min} n = \mathcal{O}(k_{\max} n \log_2 n).$$

In summary the used standard QR decompositions needs $\mathcal{O}(k_{\max}^2 n \log_2 n)$ flops.

For the orthogonalizations of block-columns against the previous columns, in summation we need not more $\mathcal{H}$-operations than for two $\mathcal{H}$-matrix-matrix multiplications. Hence the total complexity is $\mathcal{O}(k_{\max}^2 n \left(\log_2 n\right)^2)$.

■

To conclude, Algorithm 3.5 is of the same complexity as the two other $\mathcal{H}$-QR decompositions. Since $k_{\max}$ depends also on the block-wise ranks in Q and R, it is not ensured that the bound $k_{\max}$ is independent of the matrix size n.

3.3.4 Orthogonality

In this subsection we investigate the dependency of the orthogonality on the $\mathcal{H}$-arithmetic approximation error. We will use a recursive approach, starting on the lowest level to estimate $\left\|Q^TQ - I\right\|_2$. For simplification we will assume that the $\mathcal{H}$-tree is a binary tree.

On the lowest level of the $\mathcal{H}$-tree we use only the standard QR decomposition without any truncation. This is so that we can neglect the errors done on this level since the double-precision accuracy will, in general, be much higher than the precision of the $\mathcal{H}$-arithmetic approximations on the other levels.

For a binary tree the computation on level ℓ simplifies to

$$\begin{aligned}
&M = [M_1, M_2]\,,\\
&[Q_1, R_{11}] = \mathcal{H}\texttt{-QR }(M_1),\\
&R_{12} := Q_1^T *_{\mathcal{H}} M_2,\\
&\tilde{M}_2 := M_2 -_{\mathcal{H}} Q_1 *_{\mathcal{H}} R_{12},\\
&[Q_2, R_{22}] = \mathcal{H}\texttt{-QR }(\tilde{M}_2).
\end{aligned}$$

For $Q = [Q_1, Q_2]$ we want to estimate the norm of

$$Q^TQ - I = \begin{bmatrix} Q_1^TQ_1 - I & Q_1^TQ_2 \\ \left(Q_1^TQ_2\right)^T & Q_2^TQ_2 - I \end{bmatrix}.$$

From level $\ell+1$ we already know that $\left\|Q_i^TQ_i - I\right\|_2 \leq \delta_{\ell+1},\ i = 1, 2$. In order to compute $\tilde{M}_2$, two $\mathcal{H}$-matrix-matrix-multiplications are necessary. From this it follows that there is a matrix F, where $\|F\|_2 \leq 2\epsilon_\ell$, ϵ_ℓ is the $\mathcal{H}$-arithmetic approximation error on level ℓ, such that

$$\tilde{M}_2 = M_2 - Q_1Q_1^TM_2 + F.$$

For $Q_1^TQ_2$ we get

$$Q_1^TQ_2 = Q_1^T\left(M_2 - Q_1Q_1^TM_2 + F\right)R_{22}^{-1} = Q_1^TFR_{22}^{-1},$$

and the norm is

$$\left\|Q_1^TQ_2\right\|_2 \leq \|F\|_2 \left\|R_{22}^{-1}\right\|_2 \leq 2\epsilon_\ell \left\|R_{22}^{-1}\right\|_2.$$

Now we obtain the bound

$$\delta_\ell = \left\|Q^TQ - I\right\|_2 = \left\|\begin{bmatrix} Q_1^TQ_1 - I & Q_1^TQ_2 \\ \left(Q_1^TQ_2\right)^T & Q_2^TQ_2 - I \end{bmatrix}\right\|_2 \leq \sqrt{\delta_{\ell+1}^2 + 4\epsilon_\ell^2 \left\|R_{22}^{-1}\right\|_2^2}$$

on level ℓ. It follows that

$$\delta_\ell^2 \leq \delta_{\ell+1}^2 + 2c\epsilon_\ell^2,$$

if $\left\| R_{22}^{-1} \right\|_2 \leq c$. This is a strong assumption, which may not be fulfilled. For constant ϵ_ℓ we get

$$\delta_1^2 \leq \sum_{i=1}^{L} 2c\epsilon_\ell^2 = 2cL\epsilon_\ell^2.$$

If we choose $\epsilon_\ell = \epsilon \Big/ \sqrt{L}$, $r_{\mathrm{orth}}(Q)$ will be in $\mathcal{O}(\epsilon)$.

Remark 3.3.2: During the testing of this algorithm we observed that particularly the last (or the last two) column(s) of Q are not orthogonal to the previous ones. A reorthogonalization of the last column helps to get a nearly orthogonal Q.

3.3.5 Comparison to QR Decompositions for Sparse Matrices

First note that the $\mathcal{H}$-QR decomposition can be applied to dense, but data-sparse matrices arising from, for example, BEM discretizations. In such a case, sparse QR decompositions are inapplicable.

The $\mathcal{H}$-QR decomposition is also feasible for sparse matrices. Further, the data-sparse format of hierarchical matrices is related to the sparse matrix format. As such, it is useful to compare the $\mathcal{H}$-QR decomposition with the QR decomposition of sparse matrices.

The sparse QR decomposition of a large sparse matrix M leads in general to dense matrices Q and R. The same happens for the approximations $Q_\mathcal{H}$ and $R_\mathcal{H}$, to the exact Q and R we computed by the $\mathcal{H}$-QR decomposition. There are $\mathcal{O}(n^2)$ pairs $(i,j) \in I \times I$ with:

$$(Q_\mathcal{H})_{i,j} \neq 0 \quad \text{or} \quad (R_\mathcal{H})_{i,j} \neq 0.$$

But we only need a linear-polylogarithmic amount of storage to store the approximations in the $\mathcal{H}$-matrix format. In the LU decomposition there is also an analog fill-in problem, meaning that the sparsity is not preserved. The hierarchical LU decomposition is of linear-polylogarithmic complexity, too.

Some sparse QR decompositions only compute the matrix R as in [95]. This reduces the required storage and the required CPU-time. Omitting the computation of Q may be a way to accelerate the QR decomposition of $\mathcal{H}$-matrices, too. Lintner's $\mathcal{H}$-QR decomposition is most suitable in this situation, since the R is computed first, without computing Q. The sparse QR decomposition discussed in [95] uses pivoting to improve the sparsity pattern of R. Since pivoting destroys the proper chosen structure of the indices, we do not use pivoting in the $\mathcal{H}$-QR decomposition.

In the next section a comparison between the different $\mathcal{H}$-QR decompositions are given by example.

3.4 Numerical Results

We use the FEM example series, a sparse matrix with large rank-zero blocks, from FEM4 with dimension 16×16 to FEM512 with dimension $262\,144 \times 262\,144$ and the BEM example series, which is dense and without rank-zero blocks, from BEM4 (dimension 66×66) to BEM128 (dimension $65\,538 \times 65\,538$). Further, we test the algorithms with the $H_\ell(1)$ series.

We will use the accuracy and the orthogonality of the $\mathcal{H}$-QR decompositions and the required CPU-time to compare the different algorithms. Some of our test matrices are too large to store them in a dense matrix-format, so we will use

$$\begin{aligned} r_{\text{QR}}^{\mathcal{H}} &:= \|Q *_{\mathcal{H}} R -_{\mathcal{H}} M\|_2^{\mathcal{H}} \qquad \text{and} \\ r_{\text{orth}}^{\mathcal{H}}(Q) &:= \left\|Q^T *_{\mathcal{H}} Q -_{\mathcal{H}} I\right\|_2^{\mathcal{H}} \end{aligned}$$

instead of the norms in (3.1) and (3.2) to measure the orthogonality and accuracy of the $\mathcal{H}$-QR decompositions. If we compute norms using $\mathcal{H}$-arithmetic, the accuracy of the computation is determined by the approximation error of the $\mathcal{H}$-operations. If $r_{\text{QR}}^{\mathcal{H}}$ or $r_{\text{orth}}^{\mathcal{H}}$ is smaller than 10^{-5}, the $\mathcal{H}$-arithmetic pretends an accuracy which Q and R may not have. All we can say in this case is that the accuracy or the orthogonality is in the range of the approximation error or lower. In Figures 3.4 and 3.6 we plot $\max\{r_{\text{QR}}^{\mathcal{H}}, 10^{-5}\}$ and $\max\{r_{\text{orth}}^{\mathcal{H}}, 10^{-5}\}$.

The computations were done on a single core of the cluster Otto, see Section 2.7 for details. The RAM was large enough to store all matrices in the $\mathcal{H}$-matrix format. In the following, we comment on the obtained results as displayed in the Figures 3.3–3.8.

3.4.1 Lintner's $\mathcal{H}$-QR decomposition

The three different types of Lintner's $\mathcal{H}$-QR decomposition show very different behavior. The version used in Algorithm 3.1 is the fastest $\mathcal{H}$-QR decomposition in our test, but the matrices Q are far from being orthogonal, especially for large matrices. This is the only method which is faster than the $\mathcal{H}$-inversion for the FEM matrices. This method includes an $\mathcal{H}$-Cholesky factorization, meaning that it is quite natural for this method to be slower than an $\mathcal{H}$-Cholesky factorization. We should not use this method if we are interested in a factorization of our matrix as the $\mathcal{H}$-LU or $\mathcal{H}$-Cholesky factorization are better for these purposes. If we are interested in an orthogonal factorization, this method is also not the best one, since Q is often far from being orthogonal.

The second type, which uses the same method to reorthogonalize the result, is good in the two categories of orthogonality and accuracy, as long as the reorthogonalization

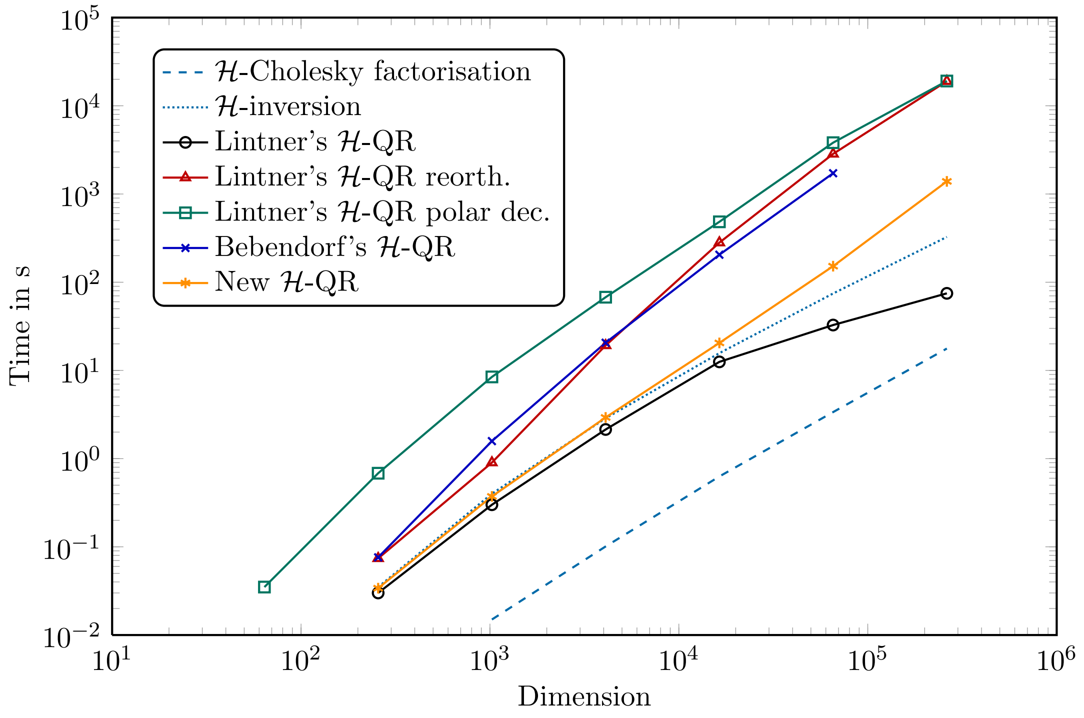


Figure 3.3: Computation time for the FEM example series.

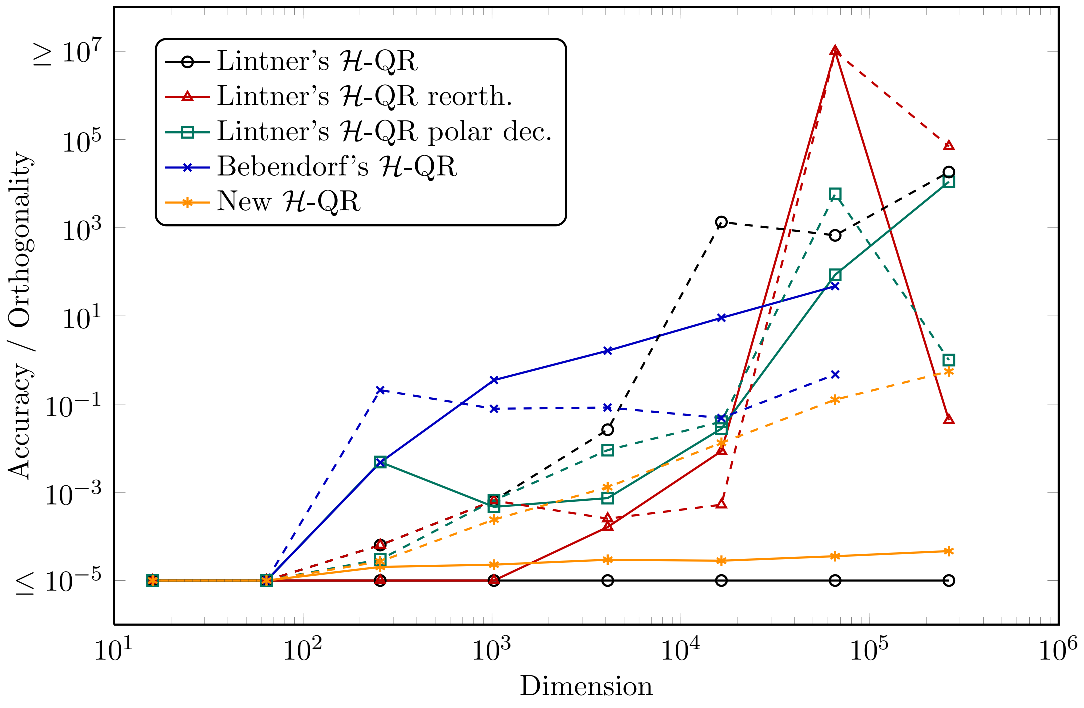


Figure 3.4: Accuracy (solid) and orthogonality (dotted) for the FEM example series.

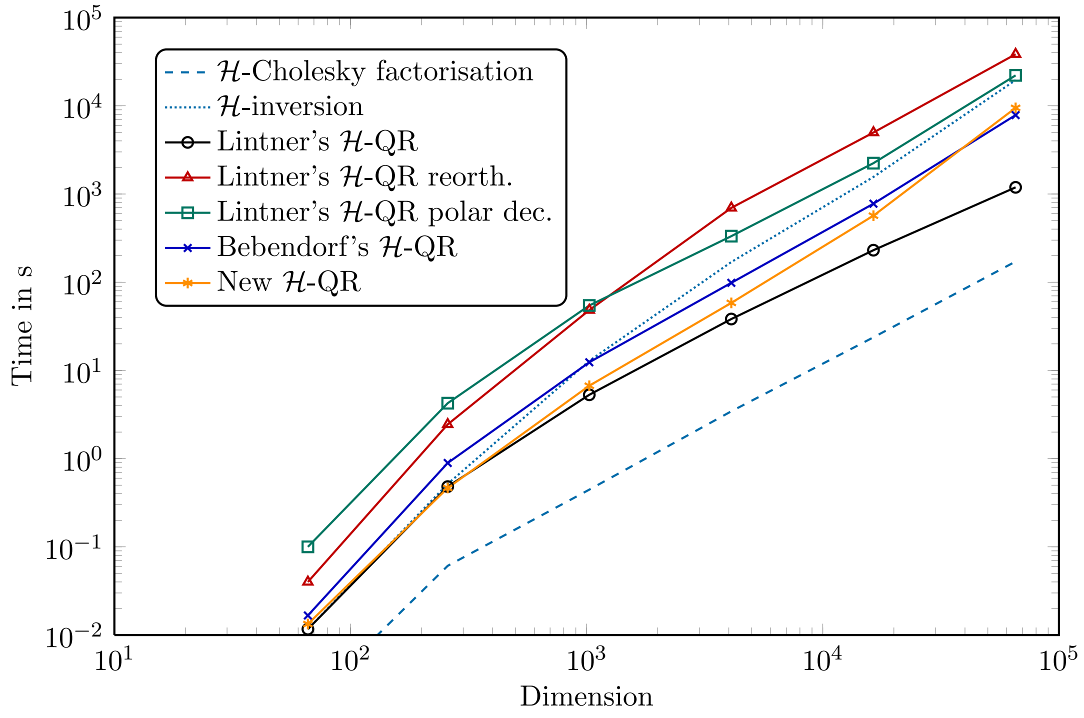


Figure 3.5: Computation time for the BEM example series.

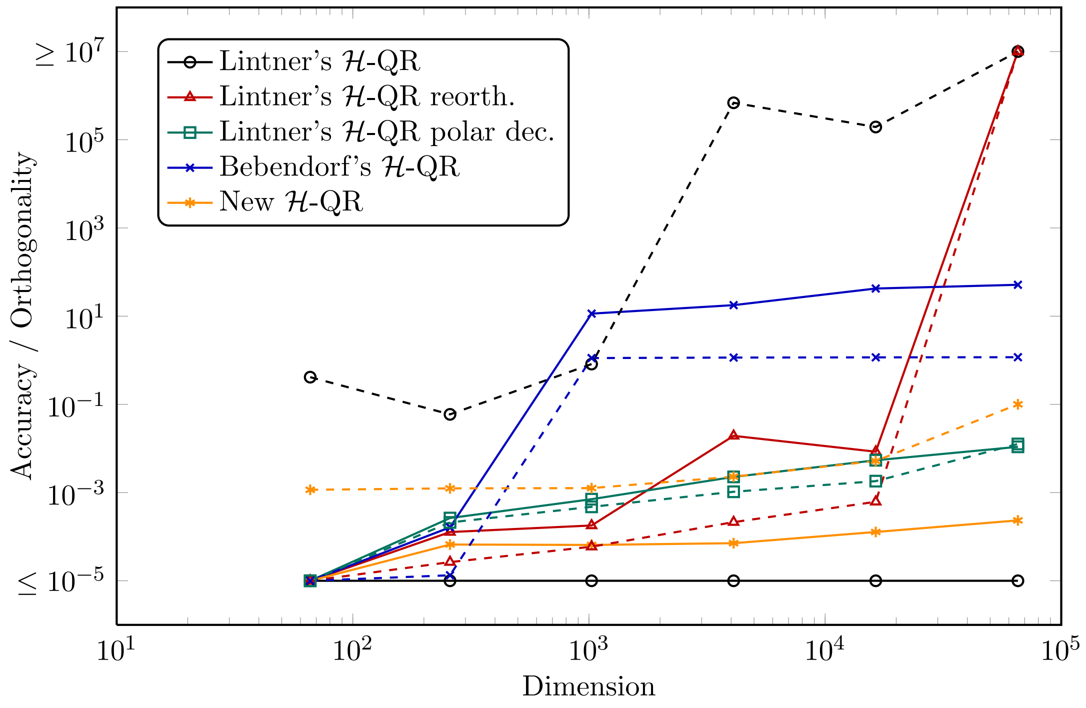


Figure 3.6: Accuracy (solid) and orthogonality (dotted) for the BEM example series.

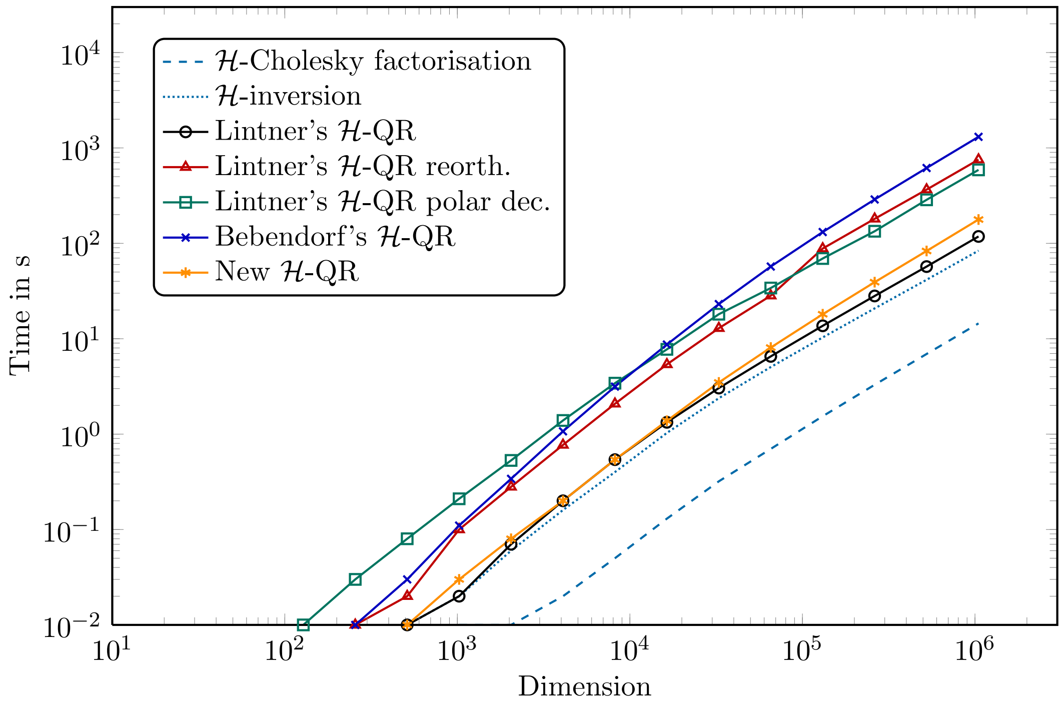


Figure 3.7: Computation time for the $\mathcal{H}_{\ell}(1)$ example series.

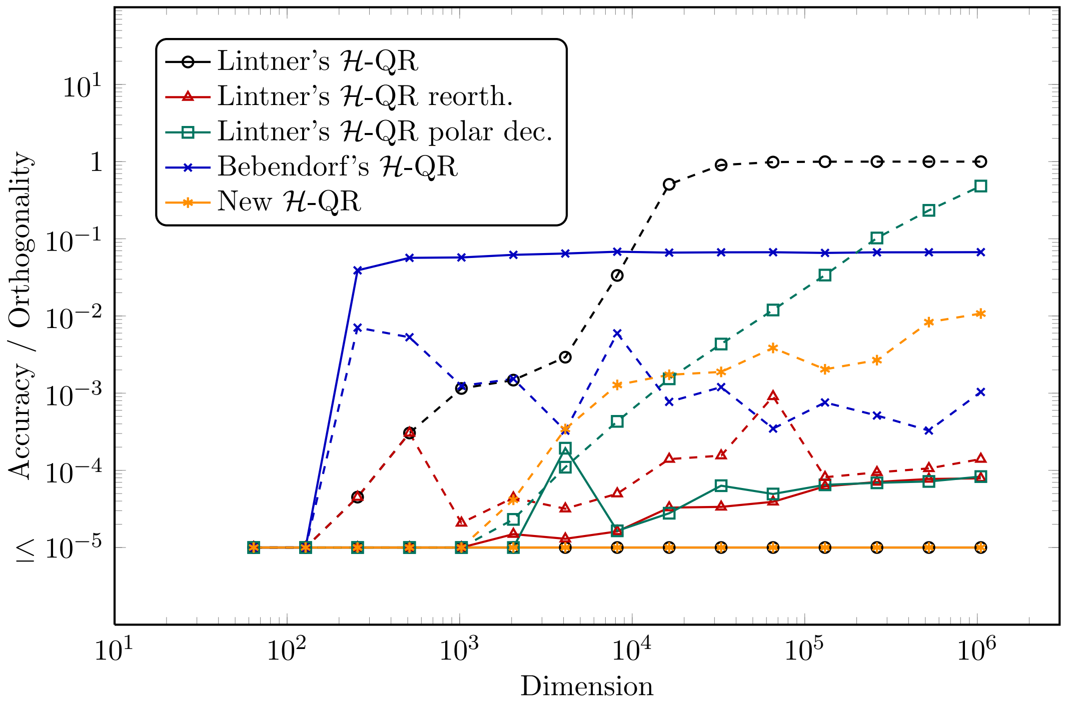


Figure 3.8: Accuracy (solid) and orthogonality (dotted) for the $\mathcal{H}_{\ell}(1)$ example series.

process converges quickly. For the shown numerical tests we stop if $r_{\text{orth}}^{\mathcal{H}} < 10^{-3}$ and if we have reached the sixth step. For the matrices FEM256, FEM512, and BEM128 the reorthogonalization does not converge within these six steps and more than six steps make the method definitely far more expensive than the others.

The third type of Lintner's $\mathcal{H}$-QR decomposition is the most or second most expensive $\mathcal{H}$-QR decomposition in this test. The orthogonality and the accuracy ranks are in the middle of the other algorithms.

3.4.2 Bebendorf's $\mathcal{H}$-QR decomposition

Algorithm 3.3 is cheaper than the last two algorithms, but the results are not as accurate and the orthogonality is not as good. The algorithm has problems with preserving the regularity of the submatrices during the recursion, and so the $\mathcal{H}$-QR decomposition of FEM512 failed. We have not tested the algorithm with a block Hessenberg matrix, but we expect better results for this type of matrices, see Remark 3.2.2.

3.4.3 The new $\mathcal{H}$-QR decomposition

Algorithm 3.5 performs best for the FEM example series. Only Lintner's reorthogonalization method attains better orthogonality, but it is not as accurate and about three times slower.

The results for the BEM matrices are not as good. For large matrices the time consumption increases too fast, making this method expensive for the largest BEM matrix. The two other indicators are good. The accuracy is as good as for the FEM matrices and the orthogonality is second or third best.

The simple structured $\mathcal{H}_\ell(1)$ matrices are more suitable for the new $\mathcal{H}$-QR decomposition. The accuracy is almost perfect and the computation takes only a little longer than the $\mathcal{H}$-inversion. The orthogonality is also the third best.

Summarizing the numerical tests, there is no method dominating all others. Some methods are good for special matrices, like Algorithm 3.3 for block Hessenberg matrices, or Algorithm 3.5 for matrices with many large zero blocks. All but Algorithm 3.1 are at least as expensive as the $\mathcal{H}$-inversion. As such, one should note:

Remark 3.4.1: The $\mathcal{H}$-QR decomposition should not be used as a preconditioner or a solver for linear systems of equations, as either Q is not orthogonal or the $\mathcal{H}$-LU/ $\mathcal{H}$-Cholesky factorization or even the $\mathcal{H}$-inversion are faster.

3.5 Conclusions

We have discussed three different methods to compute an approximate QR decomposition of $\mathcal{H}$-matrices. Such $\mathcal{H}$-QR decompositions have been suggested previously in the literature. As the known approaches have some deficiencies, either in efficiency or orthogonality of the Q-factor, we have derived a new method to compute an $\mathcal{H}$-QR decomposition. The new approach is not superior in all aspects, but offers a good compromise of accuracy vs. efficiency. We have compared the three methods in three typical sets of $\mathcal{H}$-matrices and highlighted advantages and disadvantages of the three approaches using these examples. We believe the experiments show that our approach to compute $\mathcal{H}$-QR decompositions presents a viable alternative to the existing ones. As none of the methods turns out to dominate the others' w.r.t. overall performance, we hope the presented examples help to choose the suitable $\mathcal{H}$-QR decomposition for concrete problems.

In the next chapter, the LR algorithm and the QR algorithm for hierarchical matrices are investigated. Here we apply the $\mathcal{H}$-QR decomposition.

CHAPTER

FOUR

QR-LIKE ALGORITHMS FOR HIERARCHICAL MATRICES

Contents

4.1 Introduction

In the last chapter we have seen how we can compute the QR decomposition of a general $\mathcal{H}$-matrix. In this section we will use this algorithm to build an eigenvalue algorithm, based on the explicit QR algorithm [41]. Further, we will use the related LR Cholesky algorithm [90] for the symmetric eigenvalue problem. In the following subsections we will describe both algorithms in the dense matrix case, before we apply them to hierarchical matrices in the next section.

4.1.1 LR Cholesky Algorithm

In the 1950s H. Rutishauser investigated the numerical computation of eigenvalues of matrices. Based on the qd algorithm that he published in [89], in [90] he presented the LR Cholesky algorithm. A description of Rutishauser's contributions to the solution of the algebraic eigenvalue problem can be found in the recently published paper [53]. The LR algorithm uses the following iteration:

$$\begin{aligned} L_{i+1}L_{i+1}^T &= M_i - \mu_i I \\ M_{i+1} &= L_{i+1}^{-1} M_i L_{i+1} = L_{i+1}^{-1}\left(L_{i+1}L_{i+1}^T + \mu_i I\right) L_{i+1} = L_{i+1}^T L_{i+1} + \mu_i I, \end{aligned} \tag{4.1}$$

where $L_{i+1}L_{i+1}^T$ is the Cholesky decomposition of M. One has to take care that the shifts μ_i are chosen properly, so that the shifted matrices are positive definite. More complicated shifting strategies can be found in [91] or [107], but we will use a simple shift strategy, see Subsection 4.2.2, here.

The sequence of matrices, M_i, converges to a diagonal matrix, with the eigenvalues on the diagonal as shown in [107].

4.1.2 QR Algorithm

The LR Cholesky algorithm is the predecessor of the explicit QR algorithm [41]. One simply replaces the Cholesky decomposition by the QR decomposition:

$$\begin{aligned} Q_{i+1}R_{i+1} &= M_i - \mu_i I \\ M_{i+1} &= Q_{i+1}^T M_i Q_{i+1} = Q_{i+1}^{-1}\left(Q_{i+1}R_{i+1} + \mu_i I\right) Q_{i+1} = R_{i+1}Q_{i+1} + \mu_i I. \end{aligned} \tag{4.2}$$

As a shifting strategy, one often uses the Wilkinson shift [107], the smallest eigenvalue of the trailing 2×2 submatrix. This shift does not preserve positive definiteness, but this is unnecessary for the QR decomposition.

In the QR algorithm the matrices M_i converge to an upper triangular matrix. Since each step is a unitary transformation, the eigenvalues can be found on the diagonal. In the unsymmetric case complex eigenvalues lead to 2×2 blocks on the diagonal. The eigenvalues can be found by computing the eigenvalues of all 1×1 and 2×2 blocks on the diagonal. The matrices converge to a quasi upper triangular matrix.

This also works for many other decompositions of the form $M = GR$, [106, 10]. The matrix G has to be non-singular and R an upper-triangular matrix. The algorithm of the form:

$$\begin{aligned} G_{i+1}R_{i+1} &= f_i(M_i), \\ M_{i+1} &= G_{i+1}^{-1} M_i G_{i+1}, \end{aligned}$$

is called GR algorithm, driven by f, see [105]. The functions f_i are used to accelerate the convergence. For instance $f_i(M) = M - \mu_i I$ is used in the single shift iteration and $f_i(M) = (M - \mu_{i,1}I)(M - \mu_{i,2}I) \cdots (M - \mu_{i,d}I)$ yields a multiple-shift strategy.

Sometimes the Cholesky decomposition is replaced by the LDU decomposition, too. One can show that one step of the QR algorithm is equivalent to two steps of the LR Cholesky algorithm respective LDU transformation, see [110].

4.1.3 Complexity

The QR algorithm for a dense matrix is performed on upper Hessenberg matrices to reduce the costs of the QR decompositions to $\mathcal{O}(n^2)$ flops. Further, one uses a deflation strategy to remove found eigenvalues. Together with a good shifting strategy, typically, $\mathcal{O}(n)$ iterations are sufficient to find all n eigenvalues. This leads to $\mathcal{O}(n^3)$ for the whole algorithm. Francis [42] and others improved the explicit QR algorithm to the implicit multishift QR algorithm, which is widely used today for eigenvalue computations.

If the LR Cholesky algorithm is used to compute the eigenvalues of a tridiagonal matrix, then each step costs only $\mathcal{O}(n)$ flops and the whole algorithm is of quadratic complexity. The LR Cholesky algorithm for tridiagonal matrices is efficient, since the special tridiagonal structure is exploited and preserved.

In recent years, QR-like algorithms have been developed for computing the eigenvalues of semiseparable matrices [39, 35, 36, 101, 102]. The usage of the data-sparse structure of semiseparable matrices permits the factorization in $\mathcal{O}(n)$ and so leads to algorithms of quadratic complexity, too. We are able to use some of these ideas due to the relation between semiseparable matrices and $\mathcal{H}$-matrices.

In the next section we apply the LR Cholesky algorithm to hierarchical matrices. This leads to a new eigenvalue algorithm for hierarchical matrices. Since one Cholesky decomposition is of linear-polylogarithmic complexity, we expect to find an algorithm of almost quadratic complexity. We will, however, see in the numerical results that this is not the case. This behavior will be explained in the third section, before we conclude this chapter.

Algorithm 4.1: $\mathcal{H}$-LR Cholesky Algorithm.

```
   Input: $M \in \mathbb{R}^{n\times n}$
   Output: $\Lambda \approx \Lambda(M)$
 1 Function $[\Lambda] = \mathcal{H}$-LR Cholesky algorithm($M$)
 2 begin
 3     while No deflation do
 4         Compute shift $\mu$;
 5         $L :=\mathcal{H}$-Cholesky-decomposition($M - \mu I$);
 6         $M = L^T *_{\mathcal{H}} L + \mu I$;
 7         if $\|M_{n,1:n-1}\| < \epsilon$ then
 8             $\Lambda := M_{n,n} \cup \mathcal{H}$-LR Cholesky algorithm($M_{1:n-1,1:n-1}$);
 9             break;
10         end
11         if $\exists j : \|M_{j:n,1:j-1}\| < \epsilon$ then      /* use test from Section 4.2.3 */
12             $\Lambda :=\mathcal{H}$-LR Cholesky algorithm($M_{1:j-1,1:j-1}$)$\cup$
13                             $\cup\ \mathcal{H}$-LR Cholesky algorithm($M_{j:n,j:n}$);
14         end
15     end
16 end
```

4.2 LR Cholesky Algorithm for Hierarchical Matrices

4.2.1 Algorithm

In this section the implementation of the LR Cholesky algorithm for hierarchical matrices is described. We replace the Cholesky decomposition by the approximate $\mathcal{H}$-Cholesky decomposition and the multiplication by the $\mathcal{H}$-matrix-matrix multiplication. Both arithmetic operations are of almost linear complexity and so we get Algorithm 4.1. For the $\mathcal{H}$-Cholesky decomposition we require an ordering of the indices in I. We use the ordering described in Paragraph $\mathcal{H}$-Cholesky/LU Decomposition, see page 27. In the next subsections we will investigate the shift strategy and the deflation in detail.

4.2.2 Shift Strategy

The shift is necessary to improve the rate of convergence. The shift should be a value next to the smallest eigenvalue, but smaller than the smallest eigenvalue. If we make the shift too large, the Cholesky factorization will fail, leading to a new upper bound for the smallest eigenvalue. In such a case, we should lower the shift by, for example, taking the last one, and continue with the Cholesky factorization of the new shifted matrix. The following simple shift strategy worked well in our examples:

- Randomly choose a test-vector y.
- Compute $LL^T = M$.
- Compute five steps of inverse iteration (see Section 6.1.2) $y_{i+1} := L^{-T}L^{-1}\mu_i y_i$; $\mu_{i+1} = \frac{1}{\|y_{i+1}\|_2}$ to get an approximation of the smallest eigenvalue.
- Since y_5 is in the span of several eigenvectors, $\mu_5 \geq \lambda_n$, so μ_5 is too large.
- Set $\mu := \mu_5 - 1.5 \left\| M^{-1}y_5 - \frac{1}{\mu_5}y_5 \right\|_2$, to make μ small enough.

The last step subtracts one and a half times the error estimate of the inverse iteration from the eigenvalue approximation, so that the shift preserves positive definiteness often enough. The numerical results indicate that all eigenvalues are found in $\mathcal{O}(n)$ iterations.

Every shift is smaller than the smallest eigenvalue not already deflated. The effect is that, usually, the smaller eigenvalues will be computed earlier, but there is no guarantee.

4.2.3 Deflation

Besides the shifting it is important to deflate the matrix. Deflation enables us to concentrate the computations on the not already converged eigenvalues.

To find all deflation opportunities, we have to compute

$$\|M_{j:n,1:j-1}\| \quad \forall j = 1, \ldots, n.$$

Unfortunately, this is too expensive. We compute $\|M_{j:n,1:j-1}\|$ only for $j = n$, in order to deflate the last row as soon as possible. The other tests are simplified by testing if all $\mathcal{H}$-matrix blocks on the lower left hand side have local block rank 0. E.g., we deflate in the cases shown in Figure 4.1.

4.2.4 Numerical Results

In the previous subsections we described all the things one would need to implement an LR Cholesky algorithm for $\mathcal{H}$-matrices. The implementation of that algorithm uses the $\mathcal{H}$Lib [65]. Testing the algorithm with the FEM example series leads to the computation times shown in Table 4.1. The achieved accuracy is on the level of the $\mathcal{H}$-matrix accuracy ϵ times the number of iterations. That confirms our expectations. The computation time grows like n^3. Since the number of iteration is in $\mathcal{O}(n)$, the costs per iteration has to be quadratic. As such, the reason for this is that the approximation in the arithmetic operations does not prevent the local block-wise ranks from growing, see Figure 4.2. The dark green blocks are of full rank and are together of size $\mathcal{O}(n)\times\mathcal{O}(n)$. The arithmetic for the full rank blocks has cubic complexity, so that the complexity of the whole algorithm is not almost quadratic, but cubic.

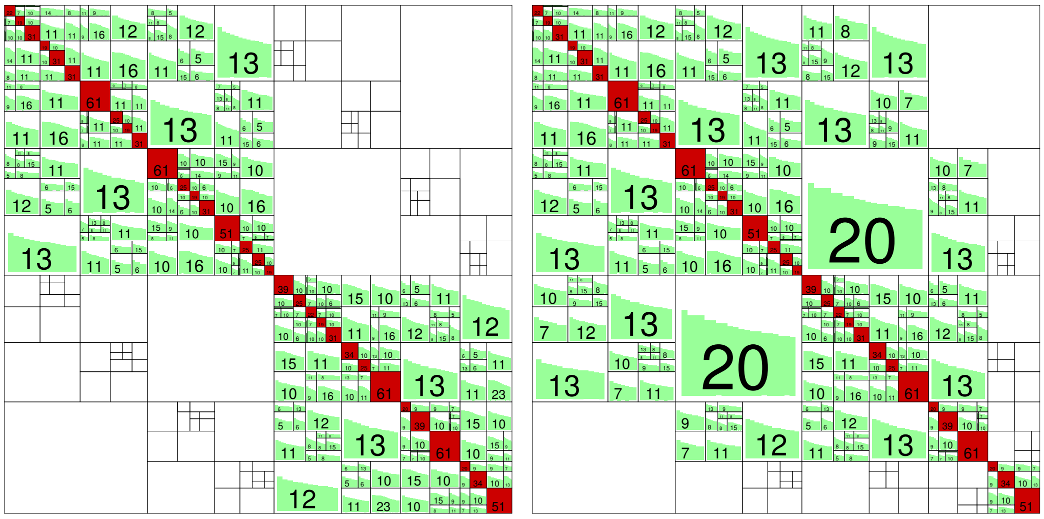


Figure 4.1: Examples for deflation.

Name	n	t_i in s	t_i/t_{i-1}	rel. error	Iterations
FEM8	64	0.05		1.5534 e−08	101
FEM16	256	4.09	82	5.2130 e−07	556
FEM32	1 024	393.73	96	5.2716 e−06	2 333
FEM64	4 096	45 757.18	116	1.6598 e−03	10 320

Table 4.1: Numerical results for the LR Cholesky algorithm applied to FEM-series.

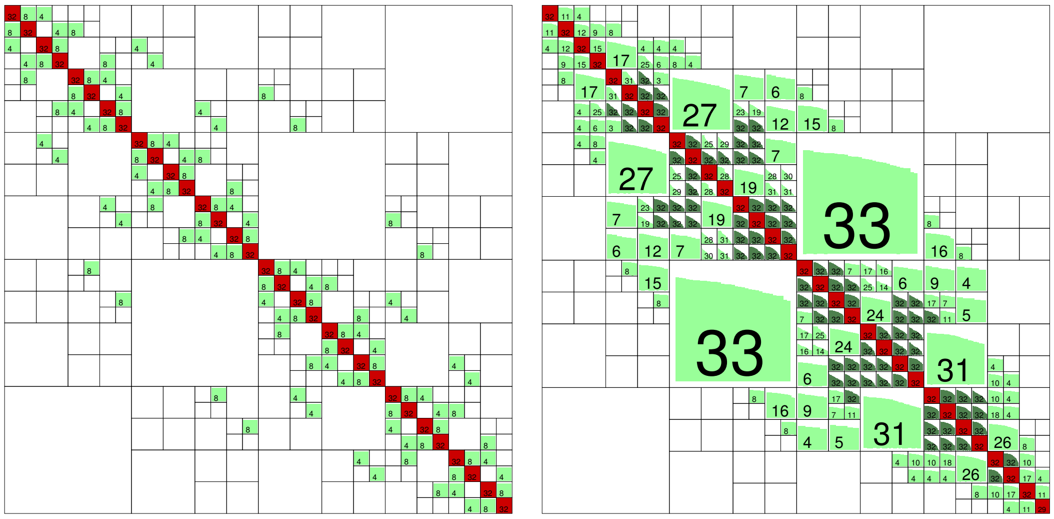


Figure 4.2: Structure of FEM32 (left) and FEM32 after 10 steps of LR Cholesky transformation (right).

Following an observation of the FEM32 example, we conclude that after a few hundred iterations the convergence towards the diagonal matrix starts to reduce the ranks of most off-diagonal blocks. Consequently, the effect of the convergence is too late, as the steps in between are too expensive.

The author implemented and tested the algorithm using the QR decomposition instead of the Cholesky factorization. Due to the equivalence between two LR Cholesky steps and one QR step, this does not lead to qualitatively different results. The only difference is that the QR algorithm does not require shifts preserving positive definiteness.

This is a disappointing result. In the next section we will give an explanation of this. Further, we will see that the algorithm can be used to compute the eigenvalues of the $\mathcal{H}_\ell$-matrices, which form a subset of $\mathcal{H}$-matrices.

4.3 LR Cholesky Algorithm for Diagonal plus Semiseparable Matrices

In this section we will explain the behavior we observed in the last section. We will give an explanation why the LR Cholesky algorithm is efficient for symmetric tridiagonal and band matrices, and is able to preserve the structure of rank structured matrices but not the structure of general symmetric hierarchical matrices. Further, we will show that a small increase in the block-wise rank is sufficient to allow the use of the LR Cholesky transformation for $\mathcal{H}_\ell$-matrices. All these matrices are diagonal plus semiseparable matrices, see Definition 2.5.2 and Remark 2.5.3, even though the structure is not storage efficient for all of them. In Subsection 4.3.4 it is explained how one can regard an $\mathcal{H}$-matrix as a semiseparable matrix with $r > n$ generators.

In the next subsection we will prove a theorem on the structure preservation of symmetric diagonal plus semiseparable matrices under LR Cholesky transformation.

4.3.1 Theorem

In this subsection we will prove that the structure of symmetric diagonal plus semiseparable matrices of rank r is preserved under LR Cholesky transformation. This is of course not new, but afterwards we will use intermediate results from our new, more constructive proof, to investigate the behavior of hierarchical structured matrices under LR Cholesky transformation. Therefore, we need the sparsity pattern of the generators of N, which is not provided by the proof in [39]. Also, the proof in [88] is insufficient to show that the preservation of structure is proved for diagonal plus semiseparable matrices in Givens-vector representation, as our matrices are in generator representation.

Theorem 4.3.1: Let M be a symmetric positive definite diagonal plus semiseparable matrix, with a decomposition like in Equation (2.24). The Cholesky factor L of

$M = LL^T$ can be written (using MATLAB notation) in the form

$$L = \operatorname{diag}\left(\tilde{d}\right) + \sum_{i=1}^{r} \operatorname{tril}\left(u^i \tilde{v}^{iT}\right). \tag{4.3}$$

Multiplying the Cholesky factors in reverse order gives the next iterate, $N = L^T L$, of the Cholesky LR algorithm. The matrix N has the same form as M,

$$N = \operatorname{diag}\left(\hat{d}\right) + \sum_{i=1}^{r} \left(\operatorname{tril}\left(\hat{u}^i \tilde{v}^{iT}\right) + \operatorname{triu}\left(\tilde{v}^i \hat{u}^{iT}\right)\right). \tag{4.4}$$

Proof. Parts of this proof are based on [88, Theorem 3.1], where a similar theorem is proved for diagonal plus semiseparable matrices in Givens-vector representation.

The diagonal entries of L fulfill:

$$\begin{aligned} L_{pp} &= \sqrt{M_{pp} - L_{p,1:p-1} L_{p,1:p-1}^T}, \\ \tilde{d}_p + \sum_i u_p^i \tilde{v}_p^i &= \sqrt{d_p + \sum_i 2 u_p^i v_p^i - L_{p,1:p-1} L_{p,1:p-1}^T}. \end{aligned} \tag{4.5}$$

This condition can easily be fulfilled when $p = 1$. Furthermore there is still some freedom for choosing $\tilde{v}_1^i$ if we choose $\tilde{d}_p$ adequately.

The entries below the diagonal fulfill:

$$\begin{aligned} L_{1:p-1,1:p-1} L_{p,1:p-1}^T &= M_{1:p-1,p} \\ L_{1:p-1,1:p-1} L_{p,1:p-1}^T &= \sum_i v_{1:p-1}^i u_p^{iT}. \end{aligned}$$

If we define $\tilde{v}_{1:p-1}^i$ by

$$L_{1:p-1,1:p-1} \tilde{v}_{1:p-1}^i = v_{1:p-1}^i, \tag{4.6}$$

then $L_{p,1:p-1} = \sum_i u_p^i \tilde{v}_{1:p-1}^{iT}$ and the above condition is fulfilled. The diagonal $\operatorname{diag}\left(\tilde{d}\right)$ results from (4.5). So the Cholesky factor has the form as in Equation (4.3).

The next iterate N is the product $L^T L$. We have

$$\begin{aligned} N &= L^T L \\ &= \left(\operatorname{diag}\left(\tilde{d}\right) + \sum_i \operatorname{tril}\left(u^i \tilde{v}^{iT}\right)\right)^T \left(\operatorname{diag}\left(\tilde{d}\right) + \sum_i \operatorname{tril}\left(u^i \tilde{v}^{iT}\right)\right) \\ &= \operatorname{diag}\left(\tilde{d}\right) \operatorname{diag}\left(\tilde{d}\right) + \sum_i \operatorname{diag}\left(\tilde{d}\right) \operatorname{tril}\left(u^i \tilde{v}^{iT}\right) + \sum_i \operatorname{tril}\left(u^i \tilde{v}^{iT}\right)^T \operatorname{diag}\left(\tilde{d}\right) + \\ &\quad + \sum_i \sum_j \operatorname{tril}\left(u^j \tilde{v}^{jT}\right)^T \operatorname{tril}\left(u^i \tilde{v}^{iT}\right). \end{aligned}$$

We will now show that $\operatorname{tril}(N,-1) = \sum_i \operatorname{tril}\left(\hat{u}^i\tilde{v}^{iT},-1\right)$:

$$\operatorname{tril}(N,-1) = \sum_i \operatorname{diag}\left(\tilde{d}\right)\operatorname{tril}\left(u^i\tilde{v}^{iT},-1\right) + \operatorname{tril}\left(\sum_{i,j}\operatorname{tril}\left(u^j\tilde{v}^{jT}\right)^T\operatorname{tril}\left(u^i\tilde{v}^{iT}\right),-1\right).$$

The other summands are zero in the lower triangular part. Define $\tilde{u}^i_p := \tilde{d}_p u^i_p$, $\forall p = 1,\dots,n$. So we get

$$\operatorname{tril}(N,-1) = \sum_i \operatorname{tril}\left(\tilde{u}^i\tilde{v}^{iT},-1\right) + \operatorname{tril}\left(\sum_i\sum_j \underbrace{\operatorname{tril}\left(u^j\tilde{v}^{jT}\right)^T\operatorname{tril}\left(u^i\tilde{v}^{iT}\right)}_{:=T^{ji}},-1\right).$$

We have $T^{ji}_{pq} = \tilde{v}^j_p u^{jT}_{p:n}u^i_{p:n}\tilde{v}^{iT}_q$, if $p > q$. It holds that

$$u^{jT}_{p:n}u^i_{p:n} = \begin{bmatrix}0 & \cdots & 0 & u^j_p & u^j_{p+1} & \cdots & u^j_n\end{bmatrix}u^i.$$

We define a matrix Z by

$$Z_{g,h} := \begin{cases}\sum_j \tilde{v}^j_g u^j_h, & g \le h,\\ 0, & g > h.\end{cases}$$

Thus,

$$Z_{p,:} = \sum_j \tilde{v}^j_p\begin{bmatrix}0 & \cdots & 0 & u^j_p & u^j_{p+1} & \cdots & u^j_n\end{bmatrix}.$$

Finally we get

$$\operatorname{tril}(N,-1) = \sum_i \operatorname{tril}\left(\underbrace{\left(\tilde{u}^i + Zu^i\right)}_{=:\hat{u}^i}\tilde{v}^{iT},-1\right) = \sum_i \operatorname{tril}\left(\hat{u}^i\tilde{v}^{iT},-1\right). \tag{4.7}$$

Since N is symmetric, the analogue holds for the upper triangle.

■

Remark 4.3.2: An analog proof for the unsymmetric case, where the semiseparable structure is preserved under LU transformations, is given in Lemma 4.5.1.

Theorem 4.3.1 tells us that $N = \operatorname{LCT}(M) := L^TL$, with the Cholesky decomposition $M = LL^T$, is the sum:

$$N = \operatorname{diag}\left(\hat{d}\right) + \sum_{i=1}^{r}\left(\operatorname{tril}\left(\hat{u}^i\tilde{v}^{iT}\right) + \operatorname{triu}\left(\tilde{v}^i\hat{u}^{iT}\right)\right),$$

with $\tilde{v}^i$ being the solution of

$$L\tilde{v}^i = v^i,$$

where L is a lower triangular matrix and

$$\hat{u}^i := \left(Z + \operatorname{diag}\left(\hat{d}\right)\right) u^i.$$

We define the set of non-zero indices of a vector $x \in \mathbb{R}^n$ by

$$\operatorname{nzi}(x) = \{i \in \{1, \ldots, n\} | x_i \neq 0\}.$$

Let i_v be the smallest index in $\operatorname{nzi}(v^i)$. Then in general $\operatorname{nzi}(\tilde{v}^i) = \{i_v, \ldots, n\}$. The sparsity pattern of $\tilde{v}^i$ is

$$\operatorname{nzi}(\tilde{v}^i) = \left\{p \in \{1, \ldots, n\} \middle| \exists q \in \operatorname{nzi}(v^i) : q \leq p\right\}.$$

Since $\tilde{u}^i_p = \tilde{d}_p u^i_p$, we have $\operatorname{nzi}(\tilde{u}^i) = \operatorname{nzi}(u^i)$. It holds that $\hat{u}^i \neq 0$ if either $\tilde{u}^i \neq 0$ or there is a j such that $\beta^{ji}_p \tilde{v}^j_p \neq 0$, with $\beta^{ji}_p = u^{jT}_{p:n} u^i_{p:n}$. The second condition is, in general, (if r is large enough) fulfilled for all $p \leq \max_{q \in \operatorname{nzi}(u^i)} q$. The sparsity pattern of $\hat{u}^i$ is

$$\operatorname{nzi}(\hat{u}^i) = \left\{p \in \{1, \ldots, n\} \middle| \exists q \in \operatorname{nzi}(u^i) : p \leq q\right\}.$$

The sparsity pattern of $\hat{u}^i$ and $\tilde{v}^i$ are visualized in Figure 4.3.

Theorem 4.3.1 shows that the structure of diagonal plus semiseparable matrices are preserved under LR Cholesky transformations. In the following subsections we will use the theorem to investigate the behavior of tridiagonal matrices, matrices with rank structures, $\mathcal{H}$-matrices and $\mathcal{H}_\ell$-matrices under LR Cholesky transformations.

$$u^i = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \star \\ \vdots \\ \star \\ 0 \\ \vdots \\ 0 \end{bmatrix} \rightsquigarrow \tilde{u}^i = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \star \\ \vdots \\ \star \\ 0 \\ \vdots \\ 0 \end{bmatrix} \rightsquigarrow \hat{u}^i = \begin{bmatrix} \star \\ \vdots \\ \star \\ \star \\ \vdots \\ \star \\ 0 \\ \vdots \\ 0 \end{bmatrix} \qquad v^i = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \star \\ \vdots \\ \star \\ 0 \\ \vdots \\ 0 \end{bmatrix} \rightsquigarrow \tilde{v}^i = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \star \\ \vdots \\ \star \\ \star \\ \vdots \\ \star \end{bmatrix}$$

Figure 4.3: Sparsity pattern of $\hat{u}^i$ and $\tilde{v}^i$.

4.3.2 Application to Tridiagonal and Band Matrices

It is well known that the band structure of a matrix is preserved under LR Cholesky transformation, as shown in [107]. The aim of this subsection is to introduce the argumentation, which we later use for hierarchically structured matrices.

Let M now be a symmetric tridiagonal matrix. This means $M_{ij} = M_{ji}$ and $M_{ji} = 0$ if $|i-j| > 1$. The matrix M can be written in the form of Equation (2.24) by setting $d = \operatorname{diag}(M)$ and

$$\begin{aligned} u^i &= M_{i+1,i} e_{i+1}, \\ v^i &= e_i. \end{aligned}$$

The matrix $N = \mathrm{LCT}(M)$ is again tridiagonal, since $\operatorname{tril}\left(\hat{u}^i \tilde{v}^{iT}\right)$ has only non-zero entries for the indices (i,i), $(i+1,i)$ and $(i+1,i+1)$.

Corollary 4.3.3: Let $M = LL^T \in \mathbb{R}^{n\times n}$ be a matrix of bandwidth b. Then $N = L^T L$ has bandwidth b, too.

Proof. Set (u^i, v^i) to

$$\begin{aligned} u^i &= \begin{bmatrix} 0 & \cdots & 0 & M^T_{i+1:i+b,i} & 0 & \cdots & 0 \end{bmatrix}^T, \\ v^i &= e_i. \end{aligned}$$

Then $\hat{u}^i_{i+b+1:n} = 0$ and $\tilde{v}^i_{1:i-1} = 0$ and so the bandwidth is preserved. ∎

4.3.3 Application to Matrices with Rank Structure

Definition 4.3.4: (rank structure) [35]
A *rank structure* on $\mathbb{R}^{n\times n}$ is a set of 4-tuples

$$\mathcal{R} = \{\mathcal{B}_k\}_k, \mathcal{B}_k = (i_k, j_k, r_k, \lambda_k).$$

If $M \in \mathbb{R}^{n\times n}$ fulfills

$$\forall k: \quad \operatorname{rank}\left((M - \lambda_k I)_{i_k:n,1:j_k} \right) \leq r_k,$$

then we call M a *matrix with rank structure* $\mathcal{R}$.

Remark 4.3.5: If M has the rank structure $(j+1, j, r, 0)$, then $M \in \mathcal{M}_{r,\{1,\ldots,j\}}$.

In [35] it is also shown that the rank structures are preserved by the QR algorithm.

Corollary 4.3.6: Let $M \in \mathbb{R}^{n \times n}$ be a matrix with rank structure $\mathcal{R} = \{\mathcal{B}_k\}_k$. Then $N = \mathrm{LCT}(M)$ has rank structure $\mathcal{R}$, too.

Proof. We can write M as diagonal-plus-semiseparable matrix. Therefore, we use

$$\mathrm{tril}\,(M - \lambda I) = \mathrm{tril}\left(\begin{bmatrix} \tilde{M}_{11} & \tilde{M}_{12} \\ AB^T & \tilde{M}_{22} \end{bmatrix}\right)$$

with the low-rank factorization $AB^T = (M - \lambda_k I)_{i_k:n,1:j_k}$ of rank at most r_k. This leads to

$$\mathrm{tril}\,(M - \lambda_k I) = \mathrm{tril}\left(\begin{bmatrix} 0 \\ A \end{bmatrix} \begin{bmatrix} B^T & 0 \end{bmatrix}\right) + \mathrm{tril}\left(\begin{bmatrix} I \\ 0 \end{bmatrix} \begin{bmatrix} \tilde{M}_{11} & \tilde{M}_{12} \end{bmatrix}\right) + \mathrm{tril}\left(\begin{bmatrix} 0 \\ I \end{bmatrix} \begin{bmatrix} 0 & \tilde{M}_{22} \end{bmatrix}\right).$$

After the LR Cholesky transformation we get:

$$\mathrm{tril}\,(N) = \mathrm{tril}\left(\begin{bmatrix} \star \\ \star \end{bmatrix} \begin{bmatrix} \star & \star \end{bmatrix}\right) + \mathrm{tril}\left(\begin{bmatrix} \star \\ 0 \end{bmatrix} \begin{bmatrix} \star & \star \end{bmatrix}\right) + \mathrm{tril}\left(\begin{bmatrix} \star \\ \star \end{bmatrix} \begin{bmatrix} 0 & \star \end{bmatrix}\right) + \mathrm{diag}\,(d)\,.$$

In the first summand, we have still a low-rank factorization of rank at most r_k. The other summands are zero in the lower left block, so that the rank structure $\mathcal{B}_k$ is preserved. This holds for all k, so that $\mathcal{R}$ is preserved, too.

■

4.3.4 Application to $\mathcal{H}$-Matrices

Let $M = M^T$ now be a symmetric $\mathcal{H}$-matrix, with a symmetric hierarchical structure. Under a symmetric hierarchical structure, we will understand that the blocks $M|_b$, where $b = s \times t$, and $M|_{b^T}$, where $b^T = t \times s$ are symmetric, so that b^T is admissible if and only if b is admissible. Further, we assume for $b \in \mathcal{L}^+$ that $M|_b = AB^T = \left(BA^T\right)^T = \left(M|_{b^T}\right)^T$, so that the ranks are equal, $k_b = k_{b^T}$. Since the Cholesky decomposition of M leads to a matrix with invertible diagonal blocks of full rank, we assume that the diagonal blocks of M are inadmissible. Further, all other blocks should be admissible. Inadmissible non-diagonal blocks will be treated as admissible blocks with full rank.

The matrix M can now be written in the form of Equation (2.24). First we rewrite the inadmissible blocks. We assume that all diagonal blocks are inadmissible and that all other blocks are admissible. We choose block $b = s \times s$ and the diagonal of $M|_b$ forms $d|_s$. For the off-diagonal entries M_{pq} and M_{qp}, we have $|p - q| \leq n_{\min}$. We need at most $n_{\min} - 1$ pairs (u^i, v^i) to represent the inadmissible block by the sum

$$\mathrm{diag}\,(d) + \sum_i \mathrm{tril}\left(u^i v^{iT}\right) + \mathrm{triu}\left(v^i u^{iT}\right).$$

We choose the first pair, so that the first columns of $u^1 v^{1T}$ and $M|_b$ coincide. The next pair $u^2 v^{2T}$ has to be chosen, so that it is equal to $M|_b - u^1 v^{1T}$ in the second column,

$$\operatorname{tril}\left(u^i v^{iT}\right) = \begin{bmatrix} 0 &&&&&&& \\ 0 & 0 &&&&&& \\ 0 & 0 & 0 &&&&& \\ 0 & 0 & 0 & 0 &&&& \\ 0 & \star & \star & 0 & 0 &&& \\ 0 & \star & \star & 0 & 0 & 0 && \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \rightsquigarrow \operatorname{tril}\left(\hat{u}^i \tilde{v}^{iT}\right) = \begin{bmatrix} 0 &&&&&&& \\ 0 & \star &&&&&& \\ 0 & \star & \star &&&&& \\ 0 & \star & \star & \star &&&& \\ 0 & \star & \star & \star & \star &&& \\ 0 & \star & \star & \star & \star & \star && \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 4.4: Example for sparsity patterns of tril $\left(u^i v^{iT}\right)$ and tril $\left(\hat{u}^i \tilde{v}^{iT}\right)$.

and so on. Like for block/band matrices, these pairs from inadmissible blocks do not cause problems to the $\mathcal{H}$-structure, since tril $\left(\hat{u}^i \tilde{v}^{iT}\right)$, with $u^i = M_{pq} e_p$ and $v^i = e_q$, has non-zero entries only in the product index set $p : q \times p : q$. This index set is part of the inadmissible block b, which contains $p \times q$. So these entries are non-zero anyway.

Further we have admissible blocks. Each admissible block in the lower triangular is a sum of products:

$$M|_b = AB^T = \sum_{j=1}^{k_b} A_{j,\cdot} \left(B_{j,\cdot}\right)^T .$$

For each pair $(A_{j,\cdot}, B_{j,\cdot})$ we introduce a pair (u^i, v^i) with $u^{iT} = \begin{bmatrix} 0 & \cdots & 0 & A_{j,\cdot}^T & 0 & \cdots & 0 \end{bmatrix}$ and $v^{iT} = \begin{bmatrix} 0 & \cdots & 0 & B_{j,\cdot}^T & 0 & \cdots & 0 \end{bmatrix}$, so that $M|_b = \sum u^i v^{iT}\big|_b$.

After we have done this for all blocks, we are left with M in the form of Equation (2.24) with, what is most likely to be, an upper summation index $r \gg n$. This means that a constant number r of pairs (u^i, v^i) under LR Cholesky transformation is insufficient for the preservation of the $\mathcal{H}$-matrix structure. Further, we require the preservation of the sparsity pattern of u^i and v^i, since otherwise the ranks of other blocks are increased (or we have to find new low rank factors, so that $\sum_j^{k_b} \tilde{A}_{j,\cdot} \tilde{B}_{j,\cdot}^T = \sum_{j=1}^{k} u^j v^{jT}$). Exactly this is not the case for general $\mathcal{H}$-matrices, since these pairs have a more general structure and cause larger non-zero blocks in N, as shown in Figure 4.3 for an example. The matrix M has a good $\mathcal{H}$-approximation, since for all i: tril $\left(u^i v^{iT}\right)$ has non-zeros only in one block of the $\mathcal{H}$-product tree. In the matrix N, the summand tril $\left(\hat{u}^i \tilde{v}^{iT}\right)$ has non-zeros in many blocks of the $\mathcal{H}$-product tree and we would need a rank-1 summand in each of these blocks to represent the summand tril $\left(\hat{u}^i \tilde{v}^{iT}\right)$ correctly. This means that the blocks on the upper right hand side of the original blocks have ranks increased by 1. Since this happens for many indices i, recall $r \gg n$, many blocks in N have full or almost full rank. In short, N is not representable by an $\mathcal{H}$-matrix of small local rank resp. the $\mathcal{H}$-matrix approximation property of M is not preserved under LR Cholesky transformations.

The complexity of the LR Cholesky algorithm for $\mathcal{H}$-matrices is in $\mathcal{O}(n^4)$. The required storage for N can only be bounded by $\mathcal{O}(n^2)$. The Cholesky factorization of a matrix with $\mathcal{O}(n^2)$ storage entries costs $\mathcal{O}(n^3)$ flops. Since typically $\mathcal{O}(n)$ iterations are necessary, the algorithm requires $\mathcal{O}(n^4)$ flops. This is a pessimistic bound, since we observe that there are data-sparse iterates again after a few hundred iterations. In the next subsection we will see that the structure of $\mathcal{H}_\ell$-matrices is almost preserved under LR Cholesky transformation.

4.3.5 Application to $\mathcal{H}_\ell$-Matrices

$\mathcal{H}_\ell$-matrices are $\mathcal{H}$-matrices with a simple structure. Let M be an $\mathcal{H}_\ell$-matrix of rank k. On the highest level we have the following structure:

$$M = \begin{bmatrix} M_{11} \in \mathcal{H}_{\ell-1} & BA^T \\ AB^T & M_{22} \in \mathcal{H}_{\ell-1} \end{bmatrix}.$$

Like in the previous subsection we introduce k pairs (u^i, v^i) for each dyad in AB^T. These pairs have the following structure:

$$u^{iT} = \begin{bmatrix} 0 & \cdots & 0 & \left(A|_{i,\cdot}\right)^T \end{bmatrix} \qquad \text{and} \quad v^{iT} = \begin{bmatrix} \left(B|_{i,\cdot}\right)^T & 0 & \cdots & 0 \end{bmatrix},$$

and so the sparsity patterns are

$$\hat{u}^{iT} = \begin{bmatrix} \star & \cdots & \star & \star & \cdots & \star \end{bmatrix} \qquad \text{and} \quad \tilde{v}^{iT} = \begin{bmatrix} \star & \cdots & \star & \star & \cdots & \star \end{bmatrix},$$

after the LR Cholesky transformation. Like for matrices with rank structure, the non-zeros from the diagonal blocks do not spread into the off-diagonal block of AB^T. The rank of the off-diagonal block on the highest level will be k, as shown in Figure 4.5.

The rank of the blocks on the next lower level will be increased by k due to the pairs from the highest level. By a recursion we get the rank structure from Figure 4.5.

Corollary 4.3.7: If $M \in \mathcal{H}_\ell(k)$, then $N = LCT(M) \in \mathcal{H}_\ell(\ell k)$.

Proof. One proof is given above. Another proof uses the relation between $\mathcal{H}_\ell$-matrices and matrices with rank structure. With [58, Lemma 4.5] we have that

$$M \in \mathcal{H}_\ell(k) \quad \Rightarrow \quad M \in \mathcal{M}_{\ell k,\tau}\ \forall \tau \in T_I.$$

The sets $\mathcal{M}_{\ell k,\tau}$ are special rank structured matrices, so that Corollary 4.3.6 gives us $N \in \mathcal{M}_{\ell k,\tau}$. The Remark 4.4 in [58] completes the proof. □

So the structure of $\mathcal{H}_\ell(k)$-matrices is *not* preserved under LR Cholesky transformations, but the maximal block-wise rank is bounded by ℓk. Since the smallest blocks have the largest ranks, the total storage required by the low-rank parts of M is only increased from

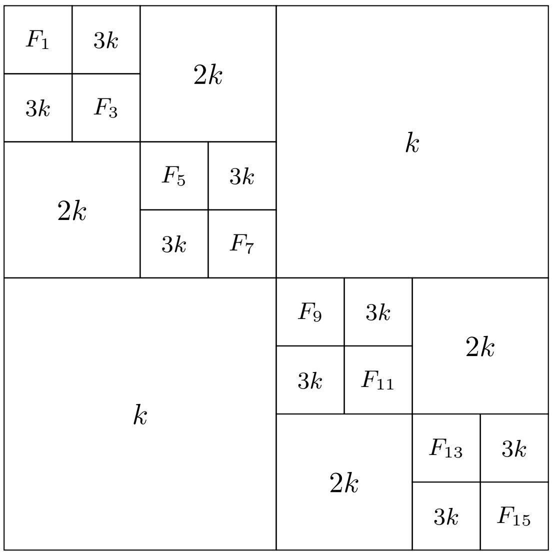


Figure 4.5: Ranks of an $\mathcal{H}_3(k)$-matrix after LR Cholesky transformation.

$2nk\ell$ to $nk\ell(\ell-1)$, meaning that the matrix has a storage complexity of $\mathcal{O}(n\,(\log_2 n)^2)$ instead of $\mathcal{O}\,(n \log_2 n)$. This is still small enough to perform an LR Cholesky algorithm in almost quadratic complexity.

If M has an additional HSS structure [109], this structure will be destroyed in the first step and M_i will be only an $\mathcal{H}_\ell$-matrix. The argumentation is analog to the next subsection on $\mathcal{H}^2$-matrices. Further the HSS structure gives only a small advantage in the computations of the first step.

4.3.6 Application to $\mathcal{H}^2$-Matrices

The $\mathcal{H}^2$-matrices are $\mathcal{H}$-matrices with the additional condition that

$$M_{r\times s} = V_r S_{r\times s} W_s^T$$

in the admissible leaves, as shown in Subsection 2.5.1. We now apply the same argumentation as before. We have

$$L\tilde{v}^i = v^i, \text{ with } v^i = \underbrace{\begin{bmatrix} 0 \\ \vdots \\ 0 \\ V_r \\ 0 \\ \vdots \\ 0 \end{bmatrix}}_{=Y_r} S_{r\times s}.$$

By computing

$$L\tilde{Y}_r = Y_r$$

the sparsity structure of Y_r is destroyed meaning that the next iterate does not have a common row span and so no $\mathcal{H}^2$-matrix structure.

4.4 Numerical Examples

In the last section it was shown that the LR Cholesky algorithm for hierarchical matrices can be used to compute the eigenvalues of $\mathcal{H}_\ell$ matrices. In this section we will test the algorithm. Therefor we use randomly generated $\mathcal{H}_\ell(1)$- and $\mathcal{H}_\ell(2)$-matrices of dimension $n = 64$ to $n = 262\,144$, with a minimum block size of $n_{\min} = 32$. Since the structure of $\mathcal{H}_\ell$-matrices is almost preserved, we expect the required CPU time for the algorithm to grow like

$$\mathcal{O}\left(k^2 n^2 \left(\log_2 n\right)^2\right).$$

In each iteration we compute a Cholesky decomposition for an $\mathcal{H}_\ell(k\ell)$-matrix, which costs $\mathcal{O}\left(k^2 n \left(\log_2 n\right)^2\right)$, see [58]. Since we expect $\mathcal{O}(n)$ iterations, we obtain the above mentioned complexity for the whole algorithm. In Figure 4.6 one can see that for the $\mathcal{H}_\ell(1)$-matrices, the CPU time grows as expected. Due to the k^2 in the complexity estimate the computations for the $\mathcal{H}_\ell(2)$-matrices are more expensive.

The graph for the CPU time of the LAPACK [2] function `dsyev` is only for comparison, as we use LAPACK 3.1.1. The CPU time of `dsyev` does not depend on the rank of the $\mathcal{H}_\ell$-matrices.

For larger examples we expect an intersection between the graphs for LAPACK function `dsyev` and for the $\mathcal{H}$-LR Cholesky algorithm. Further, due to the lower memory consumption larger matrices can be handled by the $\mathcal{H}$-LR Cholesky algorithm. Due to the large run times we have only used matrices with dimension $\leq 16\,384$.

4.5 The Unsymmetric Case

If M is not symmetric, then we must use the LU decomposition, which was called LR decomposition by Rutishauser, instead of the Cholesky decomposition. The following lemma is an analogue to Theorem 4.3.1.

Lemma 4.5.1: Let $M \in \mathbb{R}^{n\times n}$ be a diagonal plus semiseparable matrix of rank (r, s) in generator representable form:

$$M = \operatorname{diag}(d) + \sum_{j=1}^{r} \operatorname{tril}\left(u^j v^{jT}\right) + \sum_{i=1}^{s} \operatorname{triu}\left(w^i x^{iT}\right).$$

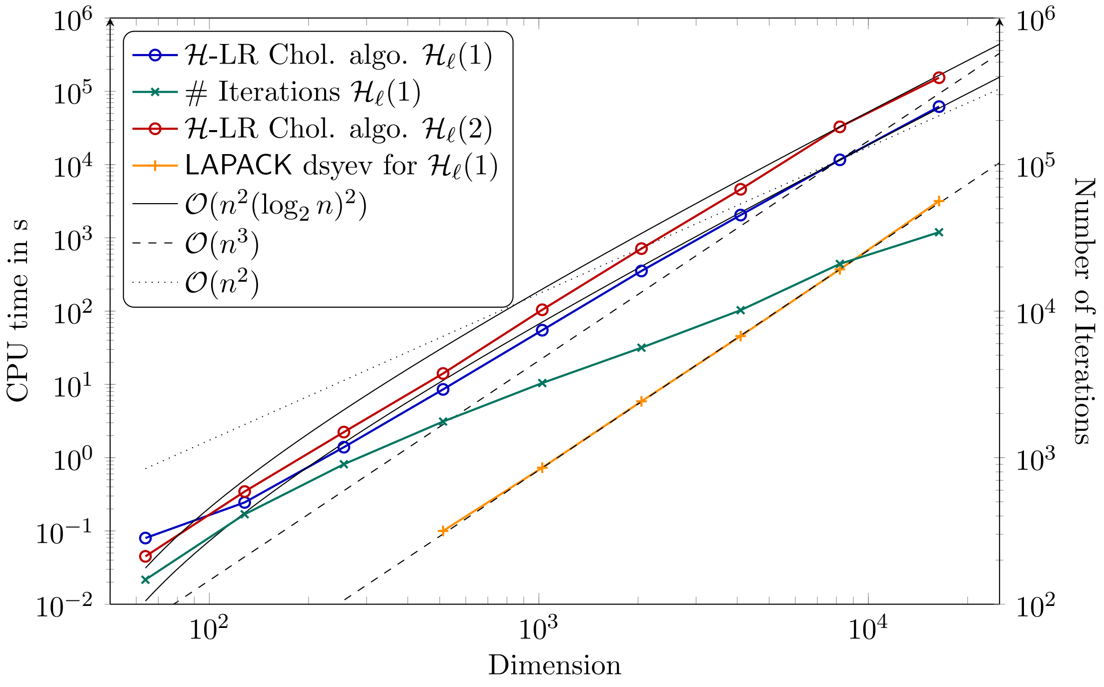


Figure 4.6: CPU time LR Cholesky algorithm for $\mathcal{H}_\ell(1)$, $n_{\min} = 32$.

Then the LU decomposition of M leads to

$$L = \operatorname{diag}\left(\tilde{d}\right) + \sum_{j=1}^{r} \operatorname{tril}\left(u^j \tilde{v}^{jT}\right),$$

$$U = \operatorname{diag}\left(\tilde{e}\right) + \sum_{i=1}^{s} \operatorname{triu}\left(\tilde{w}^i x^{iT}\right).$$

The multiplication in reverse order gives the next iterate $N = UL$ of the form

$$N = \operatorname{diag}\left(\hat{d}\right) + \sum_{j=1}^{r} \operatorname{tril}\left(\hat{u}^j \tilde{v}^{jT}\right) + \sum_{i=1}^{s} \operatorname{triu}\left(\tilde{w}^i \hat{x}^{iT}\right),$$

where r and s are unchanged.

Proof. From $M = LU$ we know that

$$L_{p,1:p-1}U_{1:p-1,1:p-1} = M_{p,1:p-1} \qquad L_{p,p} = 1 \tag{4.8}$$

$$L_{1:p-1,1:p-1}U_{p,1:p-1} = M_{1:p-1,p} \qquad U_{p,p} = M_{p,p} - L_{p,1:p-1}U_{1:p-1,p}. \tag{4.9}$$

The argumentation is now analog to the one in the proof of Theorem 4.3.1. For each p we first compute the new column of U, then the diagonal entry of the last column of U, and finally the new row of L. We assume U has the form

$$U = \operatorname{diag}(\tilde{e}) + \sum_{i=1}^{s} \operatorname{triu}\left(\tilde{w}^i \tilde{x}^{iT}\right),$$

then Equation (4.9) becomes

$$L_{1:p-1,1:p-1}\tilde{w}^i_{1:p-1}\tilde{x}^i_p = w^i_{1:p-1}x^i_p \qquad \forall i \in \{1,\ldots,s\}.$$

This equation holds for $\tilde{x}^i = x^i$ and $L\tilde{w}^i = w^i$. It can be solved up to row $p-1$, since $L_{p-1,p-1} = 1$ by definition. The equation for the diagonal entry $U_{p-1,p-1}$ is fulfilled by choosing a suitable $\tilde{e}_{p-1}$. Further, we assume L to be of the form

$$L = \operatorname{diag}\left(\tilde{d}\right) + \sum_{j=1}^{r} \operatorname{triu}\left(\tilde{u}^j \tilde{v}^{jT}\right),$$

meaning we must choose $\tilde{d}_p$ so that $L_{pp} = 1$. Further, we have to fulfill Equation (4.8), so that

$$\tilde{u}^j_p \tilde{v}^j_{1:p-1} U_{1:p-1,1:p-1} = u^j_p v^{jT}_{1:p-1}.$$

This can be achieved by setting $\tilde{u} = u$ and

$$U^T_{1:p-1,1:p-1}\tilde{v}^j_{1:p-1} = v^j_{1:p-1}.$$

So both factors have the desired form.

The next iterate is computed by

$$\begin{aligned}
N = UL &= \left(\operatorname{diag}(\tilde{e}) + \sum_{i=1}^{s}\operatorname{triu}\left(\tilde{w}^i x^{iT}\right)\right)\left(\operatorname{diag}\left(\tilde{d}\right) + \sum_{j=1}^{r}\operatorname{tril}\left(u^j \tilde{v}^j\right)\right)\\
&= \operatorname{diag}(\tilde{e})\operatorname{diag}\left(\tilde{d}\right) + \sum_{j=1}^{r}\operatorname{diag}(\tilde{e})\operatorname{tril}\left(u^j\tilde{v}^j\right) + \sum_{i=1}^{s}\operatorname{triu}\left(\tilde{w}^i x^{iT}\right)\operatorname{diag}\left(\tilde{d}\right) + \\
&\quad + \sum_{i=1}^{s}\sum_{j=1}^{r}\operatorname{triu}\left(\tilde{w}^i x^{iT}\right)\operatorname{tril}\left(u^j\tilde{v}^j\right).
\end{aligned}$$

We will now show that $\operatorname{tril}(N,-1) = \sum_{j=1}^{r}\operatorname{tril}\left(\hat{u}^j\tilde{v}^{jT}, -1\right)$:

$$\operatorname{tril}(N,-1) = \sum_{j=1}^{r}\operatorname{diag}(\tilde{e})\operatorname{tril}\left(u^j\tilde{v}^j, -1\right) + \operatorname{tril}\left(\sum_{i=1}^{s}\sum_{j=1}^{r}\underbrace{\operatorname{triu}\left(\tilde{w}^i x^{iT}\right)\operatorname{tril}\left(u^j\tilde{v}^j\right)}_{:=T^{ij}}, -1\right).$$

The other summands are zero in the lower triangular part. We have $T^{ij}_{pq} = \tilde{w}^i_p x^{iT}_{p:n} u^j_{p:n} \tilde{v}^{jT}_q$, if $p > q$. We define a matrix Z by

$$Z_{p,:} = \sum_{i=1}^{s} \tilde{w}^i_p \begin{bmatrix} 0 & \cdots & 0 & x^i_p & x^i_{p+1} & \cdots & x^i_n \end{bmatrix}$$

and get

$$\operatorname{tril}(N,-1) = \sum_{j=1}^{r} \operatorname{tril}\left(\left(\operatorname{diag}(\tilde{e})\, u^j + Z u^j\right) \tilde{v}^{jT}, -1\right) = \sum_{j=1}^{r} \operatorname{tril}\left(\hat{u}^j \tilde{v}^{jT}, -1\right).$$

We will now show with an analog argumentation that the upper triangular part is of semiseparable structure, too. We have

$$\operatorname{triu}(N,1) = \sum_{i=1}^{s} \operatorname{triu}\left(\tilde{w}^i x^i, 1\right) \operatorname{diag}\left(\tilde{d}\right) + \operatorname{triu}\left(\sum_{i=1}^{s}\sum_{j=1}^{r} \underbrace{\operatorname{triu}\left(\tilde{w}^i x^{iT}\right) \operatorname{tril}\left(u^j \tilde{v}^j\right)}_{:=T^{ij}}, 1\right).$$

The other summands are zero in the upper triangle. We have $T^{ij}_{pq} = \tilde{w}^i_p x^{iT}_{q:n} u^j_{q:n} \tilde{v}^{jT}_q$, if $q > p$. We define a matrix Y by

$$Y_{:,q} = \sum_{i=1}^{s} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ u^j_q \\ u^j_{q+1} \\ \vdots \\ u^j_n \end{bmatrix} \tilde{v}^{jT}_q .$$

Finally we get

$$\operatorname{triu}(N,1) = \sum_{i=1}^{s} \operatorname{triu}\left(\tilde{w}^i \left(x^i \operatorname{diag}\left(\tilde{d}\right) + x^{iT} Y\right), 1\right) = \sum_{j=1}^{r} \operatorname{triu}\left(\tilde{w}^i \hat{x}^{iT}, 1\right).$$

□

The result of the proof is similar to the symmetric case as in the lower triangular we get sparsity patterns like in Figure 4.4 and analog in the upper triangular part, the transposed version of Figure 4.4. This also means that for hierarchical matrices the unsymmetric LR transformations destroy the structure.

Remark 4.5.2: Without pivoting in the LR decomposition, the LR algorithm is unstable, see [107, Chapter 8]. There is no LR decomposition with pivoting for hierarchical matrices, since the pivoting would destroy the block structure. The detailed discussion of these issues is beyond the scope of this thesis. As such, even if we provide $\mathcal{O}(n^2)$ storage to run the algorithm, regardless of whether the structure is destroyed, the algorithm will not compute good approximations to the eigenvalues.

4.6 Conclusions

We have seen that the structure of diagonal plus semiseparable matrices is invariant under LR Cholesky transformation. We used this fact to show once again that rank structured matrices, especially the subsets of tridiagonal and band matrices, are invariant under LR Cholesky transformation. Besides this, we showed that a small increase of the block-wise ranks of $\mathcal{H}_\ell$-matrices is sufficient to compute the eigenvalues by an LR Cholesky algorithm. The same is true for the subset of HSS matrices.

It was not possible to disprove the opportunity of an LR Cholesky transformation for general $\mathcal{H}$-matrices, but we gave a good reason why one should expect that such an algorithm does not exist. The reason for this is that one step of the QR algorithm is equivalent to two steps of LR Cholesky transformation, the same is true for the QR algorithm. This is substantiated by a numerical example with similar results to the ones presented in Subsection 4.2.4.

If one finds a way to transform $\mathcal{H}$-matrices into $\mathcal{H}_\ell$-matrices, then one would be able to compute the eigenvalues using the LR Cholesky transformation. Also we do not use a generalized Hessenberg-form here. So we could not negate the existence of a subspace in the set of $\mathcal{H}$-matrices, in which the $\mathcal{H}$-LR Cholesky algorithm can be performed without increasing ranks.

At this point we end the investigation of QR-like algorithms for hierarchical matrices. In the next chapters we will investigate eigenvalue algorithms of a different type. The next chapter covers a slicing algorithm of high efficiency for the computation of single eigenvalues of $\mathcal{H}_\ell$-matrices. The next but one chapter is on vector iterations, especially the preconditioned inverse iteration.

CHAPTER

FIVE

SLICING THE SPECTRUM OF HIERARCHICAL MATRICES

Contents

5.1 Introduction

The aim of this chapter is to present an application of the bisection method, which was described by Beresford N. Parlett in “The Symmetric Eigenvalue Problem” [87, p. 51], for the computation of the eigenvalues of a symmetric hierarchical matrix M. He calls this process “slicing the spectrum”. The spectrum Λ of a real, symmetric matrix is contained in $\mathbb{R}$, see Lemma 2.1.10, and so the following question is well posed: How many eigenvalues $\lambda_i \in \Lambda$ are smaller than μ? We will call this number $\nu_M(\mu)$ or $\nu(M - \mu I)$. Obviously, ν_M is a function

$$\nu_M : \mathbb{R} \to \{0, \dots, n\} \subset \mathbb{N}_0 : \nu_M(\mu) = |\{\lambda \in \Lambda(M) | \lambda < \mu\}| \tag{5.1}$$

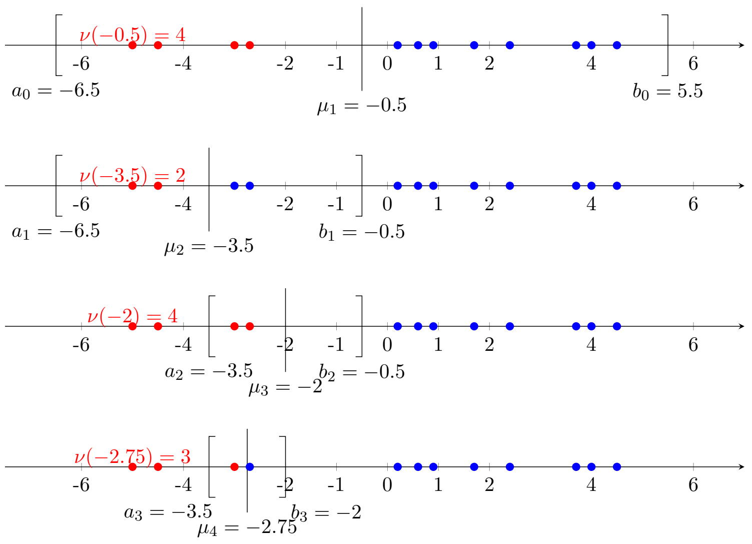


Figure 5.1: Graphical description of the bisectioning process.

If there is only one matrix M, then we omit the index M.

If the function $\nu(\cdot)$ is known, one can find the m-th eigenvalue as the limit of the following process, see Figure 5.1:

a Start with an interval $[a, b]$ with $\nu(a) < m \leq \nu(b)$.

b Determine $\nu_m := \nu(\frac{a+b}{2})$. If $\nu_m > m$, then continue with the interval $[a, \frac{a+b}{2}]$, otherwise continue with $[\frac{a+b}{2}, b]$.

c Repeat the bisection (step b) until the interval is small enough.

The function $\nu(\cdot)$ can be evaluated using the LDL^T factorization of $M - \mu I$, since Sylvester's inertia law, see Theorem 2.1.9, implies that the number of negative eigenvalues is invariant under congruence transformations. For dense matrices the evaluation of ν is expensive. Therefore, this method is not recommended if no special structure, like tridiagonality, is available.

Here we consider $\mathcal{H}_\ell$-matrices, which have such a special structure. $\mathcal{H}_\ell$-matrices can be regarded as the simplest form of $\mathcal{H}$-matrices [55]. $\mathcal{H}_\ell$-matrices include, among others, tridiagonal and numerous finite element matrices. We will see in the next section that the LDL^T factorization for $\mathcal{H}_\ell$-matrices (for all shifts) can be computed in linear-poly-logarithmic complexity. We will also show that

$$\mathcal{O}\left(k^2 n^2 (\log_2 n)^4 \log\left(\|M\|_2/\epsilon_{\text{ev}}\right)\right) \text{ flops} \tag{5.2}$$

are sufficient to find all eigenvalues with an accuracy of ϵ_{ev}, where k is the maximal rank of the admissible submatrices.

There are other eigenvalue algorithms for symmetric $\mathcal{H}_\ell$-matrices. These algorithms are described in Subsection 2.6.2 and Subsection 2.6.3 and are of quadratic-polylogarithmic complexity.

The complexity of the LDL^T slicing algorithm is competitive with the existing ones if we are interested in all eigenvalues. If we are interested only in some (interior) eigenvalues, then the algorithm will be superior, since the two others mentioned in the previous paragraph have to compute all eigenvalues. The LDL^T slicing algorithm is fundamentally different from the two other algorithms. The computational complexity depends logarithmically on the desired accuracy, so that it is really cheap to get a sketch of the eigenvalue distribution. In contrast, the algorithm can compute one eigenvalue, e.g., the smallest, second smallest, or 42nd smallest, without computing any other eigenvalue in almost linear complexity.

In the next section we will give details on how to compute all or some eigenvalues of symmetric $\mathcal{H}_\ell$-matrices. The next section is followed by a section presenting some numerical results, which demonstrates the efficiency of the algorithm. In Section 5.4 some extensions to related matrix formats are discussed. The chapter closes with a conclusions section.

5.2 Slicing the Spectrum by LDL^T Factorization

In this section the details of the slicing algorithm, mentioned in the first section, will be explained. Essentially we use a bisection method halving the interval $[a_i, b_i]$, which contains the searched eigenvalue λ_i, in each step. This process is stopped if the interval is small enough.

We will employ Algorithm 5.1 [87, p. 50ff]. If the function ν is computed exactly, the algorithm will choose the part of the interval containing λ_i. The algorithm needs $\mathcal{O}(\log_2((b-a)/\epsilon_{\text{ev}}))$ iterations to reduce the interval to size ϵ_{ev}. We know $\lambda_i \in [a_i, b_i]$, $b_i - a_i < \epsilon_{\text{ev}}$ and $\hat{\lambda}_i = (b_i + a_i)/2$ and as such it holds that

$$\left|\lambda_i - \hat{\lambda}_i\right| < \frac{1}{2}\epsilon_{\text{ev}}. \tag{5.3}$$

The evaluation of the function $\nu(\cdot)$ is the topic of the next subsection.

5.2.1 The Function $\nu(M - \mu I)$

We will need the following corollary.

Corollary 5.2.1: If $M - \mu I$ has an LDL^T factorization $M - \mu I = LDL^T$ with L invertible, then D and $M - \mu I$ are congruent and $\nu(M - \mu I) = \nu(D)$.

Algorithm 5.1: Slicing the spectrum.

```
Input: $M \in \mathcal{H}(T_{I\times I})$, with $|I| = n$ and $a, b \in \mathbb{R}$, so that $\Lambda(M) \subset [a, b]$;
Output: $\left\{\hat{\lambda}_1, \ldots, \hat{\lambda}_n\right\} \approx \Lambda(M)$;
for $i = 1, \ldots, n$ do
    $b_i := b$;   $a_i := a$;
    while $b_i - a_i \geq \epsilon_{ev}$ do
        $\mu := (b_i + a_i)/2$;
        $[L, D] :=$ LDL$^T$ factorization($M - \mu I$);
        $\nu(M - \mu I) := |\{j | D_{jj} < 0\}|$;
        if $\nu(M - \mu I) \geq i$ then $b_i := \mu$ else $a_i := \mu$ ;
    end
    $\hat{\lambda}_i := (b_i + a_i)/2$;
end
```

Proof. See Theorem 2.1.9. ∎

Since D is diagonal, we can easily count the number of positive or negative eigenvalues, which gives us $\nu(D)$ and, due to the corollary, $\nu_M(\mu)$.

If a diagonal entry of D is zero, we have shifted with an eigenvalue. In this case one of the leading principal submatrices of $M - \mu I$ is rank deficient and, as such, the LDL^T factorization may fail. Such a case is a welcome event as an eigenvalue has been found. The LDL^T factorization computes the diagonal D entry by entry. If one of these computed entries is zero, then we have detected a rank deficient principal submatrix and should stop the factorization.

We investigate the LDL^T factorization of $\mathcal{H}_\ell$-matrices in the next subsection.

5.2.2 LDL^T Factorization of $\mathcal{H}_\ell$-Matrices

There is an algorithm to compute LDL^T factorizations for hierarchical matrices, which is first described in [73, p. 70]. The $\mathcal{H}$-LDL^T factorization is block recursive, see Algorithm 5.2. This algorithm is similar to Algorithm 2.2 for the Cholesky decomposition of hierarchical matrices. For a hierarchical matrix $M \in \mathcal{H}(T, k)$ this factorization has a complexity of

$$\mathcal{O}(k^2 n (\log_2 n)^2) \tag{5.4}$$

in fixed rank $\mathcal{H}$-arithmetic. We note that the LDL^T factorization for $\mathcal{H}$-matrices is much cheaper than for dense matrices, where $\mathcal{O}(n^3)$ flops are needed. In standard arithmetic, the stability of the factorization is improved by, e.g., Bunch-Kaufmann pivoting [24].

Algorithm 5.2: $\mathcal{H}$-LDL^T factorization $M = LDL^T$.

```
1 $\mathcal{H}$-LDL$^T$ factorization($M$);
  Input: $M \in \mathcal{H}(T)$
  Output: $L \in \mathcal{H}(T)$, $D = \operatorname{diag}(d_1, \ldots, d_n)$ with $LDL^T = M$ and $L$ lower
         triangular
2 if $M = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix} \notin \mathcal{L}(T)$ then
3     $[L_{11}, D_1] := \mathcal{H}$-LDL$^T$ factorization($M_{11}$);
4     Compute the solution $L_{21}$ of $L_{21} D_1 L_{11}^T = M_{21}$;
5     $[L_{22}, D_2] := \mathcal{H}$-LDL$^T$ factorization($M_{22} - L_{21} D_1 L_{21}^T$);
6 else
7     Compute the dense LDL$^T$ factorization $LDL^T = M$, since inadmissible
      diagonal blocks are stored as dense matrices.
8 end
9 return $L, D$;
```

Pivoting can not be used here as it would destroy the hierarchical structure. Many practical problems lead to diagonally dominant matrices where in theses cases, pivoting is unnecessary for good results. Here we need only an exact evaluation of $\nu(\mu)$, and for this we do not necessarily require a highly accurate LDL^T factorization.

We will use Algorithm 5.2 for $\mathcal{H}_\ell$-matrices, too. In this case the solution of the equation

$$L_{21} D_1 L_{11}^T = M_{21}$$

is simplified, since $M_{21} = AB^T$. If D_1 has a zero entry the solution will fail. But in this case we know that zero is an eigenvalue of M. After the computation of L_{21} an update is performed. In general, this update increases the rank of the submatrix M_{22}. In fixed rank $\mathcal{H}$-arithmetic the update is followed by a truncation step, which reduces the rank again to k. For $\mathcal{H}_\ell$-matrices we will omit the truncation, since the growth of the block-wise ranks is bounded. The next lemma gives this bound, which will be used for the complexity analysis of Algorithm 5.2.

Lemma 5.2.2: Let $M \in \mathcal{H}_\ell(k)$. If the assumptions of Definition 2.1.16 are fulfilled, then the triangular matrix L of the LDL^T factorization is an $\mathcal{H}_\ell(k\ell)$-matrix. Further, the complexity of the computation of L and D by Algorithm 5.2 is

$$\mathcal{O}(k^2 n \, (\log_2 n)^4). \tag{5.5}$$

Proof. We will first prove the statement on the block-wise ranks and then use this for the complexity estimation. We number the blocks of M as in Figure 2.8, see Figure 5.2. Each block has a number out of the index set $S = \{1, \ldots, 2^{\ell+1} - 1\}$. First we will define

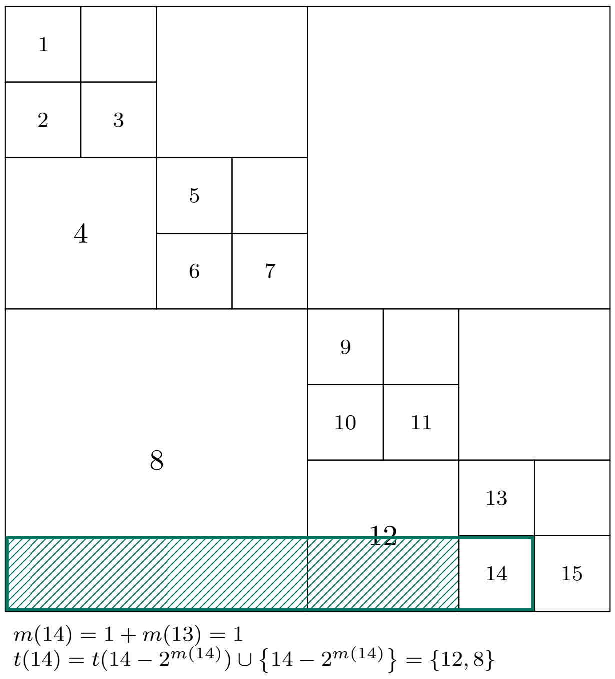

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
$m(14) = 1 + m(13) = 1$
$t(14) = t(14 - 2^{m(14)}) \cup \{14 - 2^{m(14)}\} = \{12, 8\}$

Figure 5.2: Example $\mathcal{H}_3$-matrix.

some functions on S. The function $m(i)$ is defined by

$$m(i) = \begin{cases} 0, & \text{if } i \text{ is odd,} \\ 1 + m(i/2), & \text{if } i \text{ is even.} \end{cases}$$

The size of block i is $n_0\, 2^{\max\{m(i)-1,0\}} \times n_0\, 2^{\max\{m(i)-1,0\}}$. The next function $t(i)$ gives us the indices of blocks on the left hand side of block i:

$$t(i) = \begin{cases} \emptyset, & \text{if } i - 2^{m(i)} = 0, \\ t(i - 2^{m(i)}) \cup \left\{i - 2^{m(i)}\right\}, & \text{else.} \end{cases}$$

Finally we will need

$$u(i) = |t(i)|\,.$$

Algorithm 5.2 processes the blocks in the order of their numbering. Most operations of Algorithm 5.2 do not change the block-wise ranks, only the update in line 5 increases the rank of some blocks. We are interested in the final rank of block i. Obviously only updates from blocks $j \in t(i)$ act on i. We assume that $t(i) = \left\{j_1, j_2, \ldots, j_{u(i)}\right\}$. Let the smallest index in $t(i)$ be j_1. We compute the solution of $L_{11}D_1L_{21}^T = M_{21}^T$ for block j_1. The matrix L_{21} is low rank and $L_{21} = A_{j_1} B_{L_{j_1}}$, with $B_{L_{j_1}}$ being the solution of $L_{11} D_1 B_{L_{j_1}} = B_{j_1}$. So the update $L_{21} D_1 L_{21}^T$ has the form $A_{j_1} X^T$ and is of rank k. After the update, block j_2 has the form $[A'_{j_1}, A_{j_2}][X', B_{j_2}]^T$, where $A'_{j(1)}$ and X' are the suitable parts of $A_{j(1)}$ and X. This means that the next update is of rank $2k$ and has the form $[A'_{j_1}, A_{j_2}]Y^T$. This second update increases the rank of block j_3 only by k since the block has the form $[A''_{j_1}, A_{j_3}]Z$ before the update. Finally, block i has rank $k(u(i)+1)$.

The maximum

$$\max_{i \in S} u(i) = \ell,$$

is attained only for odd i. The rank of the inadmissible diagonal blocks is not of interest, since they are stored in dense matrix format. So the maximum rank of an admissible block is bounded by $k\ell$.

The $\mathcal{H}$-LDL^T factorization does not change the hierarchical structure. As such, we have $L \in \mathcal{H}_\ell(k\ell)$. With Lemma 2.3.13, Equation (5.4) and $\ell = \mathcal{O}(\log_2 n)$, we conclude that the complexity of Algorithm 5.2 is in

$$\mathcal{O}(k^2 n \, (\log_2 n)^4).$$

■

Remark 5.2.3: $L \in \mathcal{H}_\ell(k\ell)$ can be proven in a shorter way:

Proof. The proof is analog to the proof of Lemma 2.3.19. If $M \in \mathcal{M}_{k,\tau}$, see Definition 2.3.20, then there is a permutation so that the indices are renumbered with $\tau = \{1, \ldots, m\}$, further the permuted M has the rank structure $(m+1, m, r, \cdot)$, see Definition 4.3.4.

Analog to [58, Lemma 4.2], we get for the LDL^T factorization of $M = LDL^T$, that $L \in \mathcal{M}_{k,\tau}$, if $M \in \mathcal{M}_{k,\tau}$, since

$$L|_{\tau' \times \tau} \, D_{\tau'} \, L|_{\tau' \times \tau'} = M|_{\tau' \times \tau} \, .$$

The matrices $D_{\tau'}$ and $L|_{\tau' \times \tau'}$ are of full rank, so that

$$\operatorname{rank}\left(L|_{\tau' \times \tau}\right) = \operatorname{rank}\left(M|_{\tau' \times \tau}\right).$$

Remark 4.4 and Lemma 4.5 of [58] state that

$$M \in \mathcal{M}_{k,\tau} \ \forall \tau \in T_I \Rightarrow M \in \mathcal{H}_\ell(k) \qquad \text{and}$$
$$M \in \mathcal{H}_\ell(k) \Rightarrow M \in \mathcal{H}_{p \cdot k, \tau} \ \forall \tau \in T_I^{(p)}.$$

So we get $L_\ell \in \mathcal{H}(k\ell)$.

■

Remark 5.2.4: We know from the first proof that block i has at most rank $k(u(i)+1)$. Unfortunately, this tighter rank bound does not lead to a better complexity estimation. Nevertheless, we use the tighter bound for numerical computations.

The main difference between the LDL^T factorization for $\mathcal{H}_\ell$-matrices and $\mathcal{H}$-matrices is that the factorization for $\mathcal{H}_\ell$-matrices can be done without truncation and so exact apart from round off respecting IEEE-double precision arithmetic.

Remark 5.2.5: Where M has a stronger condition, one can reduce the bound on the block-wise rank from $k\ell$ to k. If, for instance, the matrix $M \in \mathcal{H}_\ell(k)$ fulfills the following conditions (here for an $\mathcal{H}_3(k)$-matrix with the same notation as in Figure 2.8):

$$\begin{aligned}
\operatorname{range}(A_4) &\subset \operatorname{span}\left(\begin{bmatrix} F_5 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ A_6 \end{bmatrix}\right), \\
\operatorname{range}(A_8) &\subset \operatorname{span}\left(\begin{bmatrix} F_9 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ A_{10} \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ \\ A_{12} \end{bmatrix}\right) \quad \text{and} \\
\operatorname{range}(A_{12}) &\subset \operatorname{span}\left(\begin{bmatrix} F_{13} \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ A_{14} \end{bmatrix}\right),
\end{aligned}$$

or an analog generalization, then $L \in \mathcal{H}_\ell(k)$. Examples of this structure are tridiagonal matrices, diagonal plus semi-separable matrices and HSS matrices. For all these special structures there exists good eigenvalue algorithms.

5.2.3 Start-Interval $[a, b]$

The interval $[a, b]$ must contain the whole spectrum. This is the case for $a := -\|M\|_2$ and $b := \|M\|_2$. The spectral norm $\|M\|_2$ can be approximated from below using the power iteration, see Chapter 6. Multiplying the approximation by a small factor $1+\delta$ will give an upper bound for $\|M\|_2$.

5.2.4 Complexity

For each eigenvalue λ_i we have to do several $\mathcal{H}$-LDL^T factorizations to reduce the length of the interval $[a_i, b_i]$. Each factorization halves the interval, since we use a bisection method. So we need $\mathcal{O}(\log(\|M\|_2/\epsilon_{\text{ev}}))$ $\mathcal{H}$-LDL^T factorizations per eigenvalue. One $\mathcal{H}$-LDL^T has a complexity of $\mathcal{O}(k^2 n (\log_2 n)^4)$. Multiplying both complexities gives us the complexity per eigenvalue $\mathcal{O}(k^2 n (\log_2 n)^4 \log(\|M\|_2/\epsilon_{\text{ev}}))$ and the total complexity for all n eigenvalues is:

$$\mathcal{O}(k^2 n^2 (\log_2 n)^4 \log(\|M\|_2/\epsilon_{\text{ev}})). \tag{5.6}$$

5.3 Numerical Results

We have implemented Algorithm 5.1 with the LDL^T factorization for $\mathcal{H}$-matrices, as shown in Algorithm 5.2, using the $\mathcal{H}$Lib [65]. The $\mathcal{H}$Lib can handle $\mathcal{H}_\ell$-matrices, too, since $\mathcal{H}_\ell$-matrices are a subset of $\mathcal{H}$-matrices. We use the fixed rank arithmetic of the $\mathcal{H}$Lib, with the known maximal block-wise rank $k\ell$, which we get through the factorization of $\mathcal{H}_\ell(k)$-matrices. We also choose a minimum block size of $n_{\min} = 32$. The computations were done on Otto, see Section 2.7. Apart from the numerical results regarding parallelization, we use only one core.

To test the algorithm we use tridiagonal, $\mathcal{H}_\ell$-, and HSS matrices. The matrices have block-wise rank 1 in the admissible submatrices. The size of the matrices is varied from 64 to 1 048 576. Like in the $\mathcal{H}_\ell$-matrix example series, see Subsection 2.4.3, the HSS matrices are randomly generated, fulfilling the additional conditions from Definition 2.5.7. Further, we use a set of tridiagonal matrices with 2 on the diagonal and -1 on the subdiagonals, representing the discrete 1D Laplace operator. Except for the tridiagonal matrices, we normalize the matrices to $\|M\|_2 = 1$, since $\|M\|_2$ is part of the complexity estimate.

To investigate the dependency on the local block-wise rank k we use $\mathcal{H}_\ell$-matrices of dimension 16 384 with rank $2, 3, 4, 8$ or 16 (H9 r2. . . 16).

For the matrices up to dimension 32 768, we compute their corresponding dense matrix and use the LAPACK-function `dsyev` [2] to compute the eigenvalues. The difference between the results of `dsyev` and the results from our new LDL^T slicing algorithm are the errors in the tables and figures. The time `dsyev` needs is given in the table, too. Note: there are faster algorithms for tridiagonal matrices but not for $\mathcal{H}_\ell$- and HSS matrices in LAPACK.

Figure 5.3 shows the absolute errors of the computed eigenvalues of the matrix H5 r1 $\in \mathcal{H}_5(1)$ of size 1 024. All the errors are below the expected bound.

Table 5.1 shows the computation times and the errors for the Hℓ example series, if we compute only the 10 eigenvalues $\lambda_{n/4+5}, \ldots, \lambda_{n/4+14}$. (Similar results will be obtained when choosing other subsets of the spectrum.) The growth in the expected costs

$$\frac{N_i}{N_{i-1}} = \frac{n_i (\log_2 n_i)^4}{n_{i-1}(\log_2 n_{i-1})^4} \qquad \text{for constant } k \text{ and}$$

$$\frac{N_i}{N_{i-1}} = \frac{k_i^2}{k_{i-1}^2} \qquad \text{for constant } n$$

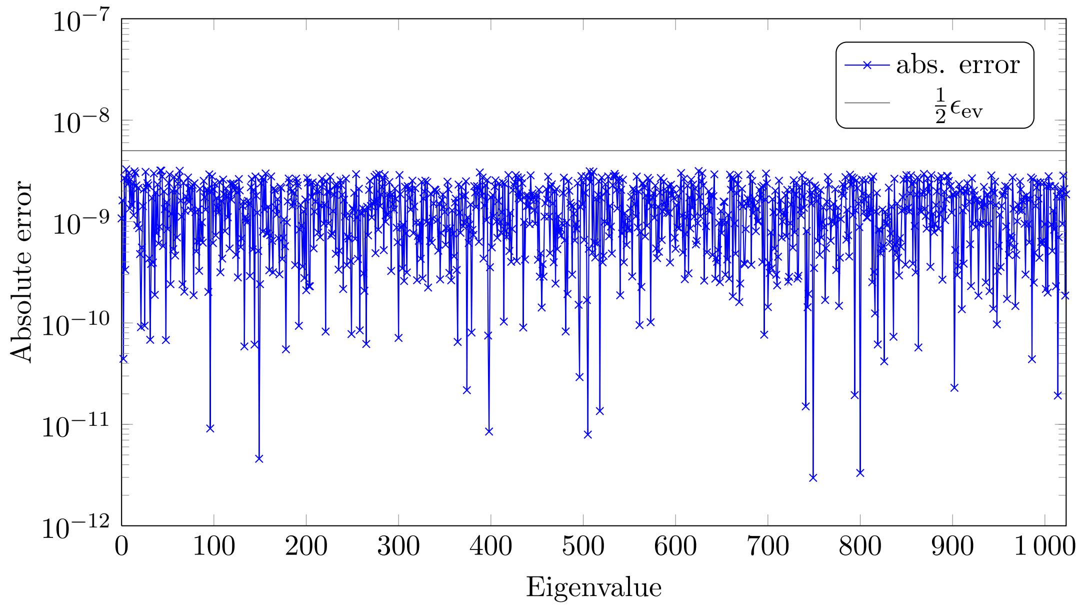


Figure 5.3: Absolute error $|\lambda_i - \hat{\lambda}_i|$ for a $1\,024 \times 1\,024$ matrix (H5 r1), $\epsilon_{\text{ev}} = 10^{-8}$.

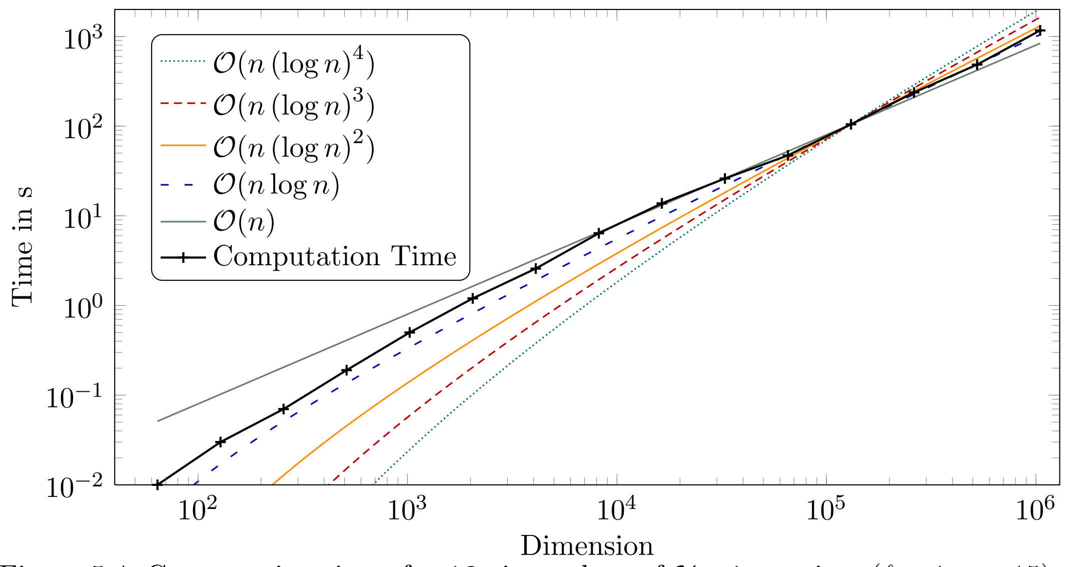


Figure 5.4: Computation times for **10** eigenvalues of $\mathcal{H}_\ell$ r1 matrices ($\ell = 1, \ldots, 15$).

Name	n	t_{dysev} in s	abs. error	rel. error	t in s	$\frac{t_i}{t_{i-1}}$	$\frac{N_i}{N_{i-1}}$
H1 r1	64	<0.01	2.6176 e−09	4.2210 e−09	0.01		
H2 r1	128	<0.01	2.6782 e−09	4.7816 e−09	0.03	3.00	3.71
H3 r1	256	0.01	3.0051 e−09	5.6233 e−09	0.07	2.33	3.41
H4 r1	512	0.09	3.0835 e−09	6.0051 e−09	0.19	2.71	3.20
H5 r1	1 024	0.65	3.0225 e−09	5.9838 e−09	0.50	2.63	3.05
H6 r1	2 048	4.96	3.2310 e−09	6.4253 e−09	1.20	2.40	2.93
H7 r1	4 096	40.04	2.4008 e−09	4.7873 e−09	2.58	2.15	2.83
H8 r1	8 192	318.41	2.2674 e−09	4.5241 e−09	6.39	2.48	2.75
H9 r1	16 384	2 578.40	2.8512 e−09	5.6983 e−09	13.76	2.15	2.69
H10 r1	32 768	21 544.30	2.2013 e−09	4.4015 e−09	26.06	1.89	2.64
H11 r1	65 536	—	—	—	47.18	1.81	2.59
H12 r1	131 072	—	—	—	104.80	2.22	2.55
H13 r1	262 144	—	—	—	237.39	2.27	2.51
H14 r1	524 288	—	—	—	485.45	2.04	2.48
H15 r1	1 048 576	—	—	—	1 167.69	2.41	2.46
H9 r1	16 384	2 578.40	2.8512 e−09	5.6983 e−09	13.58		
H9 r2	16 384	2 623.36	2.9862 e−09	5.9585 e−09	36.26	2.67	4.00
H9 r3	16 384	2 813.56	3.2523 e−09	6.5293 e−09	68.73	1.90	2.25
H9 r4	16 384	2 569.13	2.9416 e−09	5.8803 e−09	108.63	1.58	1.78
H9 r8	16 384	2 622.41	2.9974 e−09	5.9699 e−09	345.52	3.18	4.00
H9 r16	16 384	2 574.00	2.5739 e−09	5.2128 e−09	998.93	2.89	4.00

Table 5.1: Comparison of errors and computation times for the $\mathcal{H}_\ell$ example series computing only 10 eigenvalues $(n/4+5,\ldots,n/4+14)$.

Name	n	t_{dsyev} in s	abs. error	rel. error	t in s	$\frac{t_i}{t_{i-1}}$	$\frac{N_i}{N_{i-1}}$
H1 r1	64	<0.01	2.9831 e−09	7.5664 e−09	0.06		
H2 r1	128	<0.01	3.5112 e−09	7.6290 e−09	0.32	5.33	7.41
H3 r1	256	0.01	2.8642 e−09	1.7238 e−08	1.78	5.56	6.82
H4 r1	512	0.09	4.2852 e−09	5.2780 e−08	9.40	5.28	6.41
H5 r1	1 024	0.65	3.2705 e−09	3.6564 e−08	46.44	4.94	6.10
H6 r1	2 048	4.96	3.8801 e−09	2.3364 e−07	219.13	4.72	5.86
H7 r1	4 096	40.04	4.5528 e−09	3.6158 e−07	991.74	4.53	5.67
H8 r1	8 192	318.41	4.5752 e−09	4.8727 e−07	4 001.82	4.04	5.51
H9 r1	16 384	2 578.40	4.5128 e−09	1.0878 e−07	15 727.87	3.93	5.38
H10 r1	32 768	21 544.30	4.5128 e−09	4.1400 e−06	48 878.33	3.11	5.27
H11 r1	65 536	—	—	—	139 384.10	2.85	5.18
H9 r1	16 384	2 578.40	4.5872 e−09	1.0878 e−07	15 520.59		
H9 r2	16 384	2 623.36	4.5125 e−09	2.2098 e−06	43 553.35	2.81	4.00
H9 r3	16 384	2 813.56	4.4878 e−09	5.7552 e−07	90 788.42	2.08	2.25
H9 r4	16 384	2 569.13	4.6128 e−09	7.6893 e−07	141 035.10	1.55	1.78
H9 r8	16 384	2 622.41	4.5900 e−09	2.9278 e−06	459 240.80	3.26	4.00
H9 r16	16 384	2 574.00	4.6000 e−09	1.0670 e−07	1 375 369.00	2.99	4.00

Table 5.2: Comparison of errors and computation times for the $\mathcal{H}_\ell$ example series computing all eigenvalues.

is given in the last column, k and $\|M\|_2$ are constant in the first part of the table. The computation times grow slower than expected. This confirms the estimated computational complexity from Equation (5.5) and shows that there is probably a tighter bound. Figure 5.4 compares the computation times with $\mathcal{O}(n\,(\log_2 n)^\beta)$, $\beta = 0, 1, 2, 3, 4$. There we see that the β in the example is rather 2 than 4. The second part of the table shows that for varying k, the costs grow like k^2. Tables 5.2 shows the same as Tables 5.1 but for computing all eigenvalues, with $N_i = k_i^2 n_i^2 (\log_2 n_i)^4$. Table 5.2 shows that the computation of all eigenvalues with the LDL^T slicing algorithm is more expensive than using LAPACK. But since the transformation into a dense matrix requires n^2 storage, we are able to solve much larger problems by using the LDL^T slicing algorithm.

In Table 5.3, Table 5.4 and Table 5.5, similar results are shown for tridiagonal, HSS matrices and the $\mathcal{H}_\ell(5)$-matrices with the kernel function $\log \|x - y\|$.

5.4 Possible Extensions

In the last two sections we have described an algorithm to compute the eigenvalues of $\mathcal{H}_\ell$-matrices. In this section we will discuss what happens if we apply this algorithm to other, related hierarchically structured matrices. Further, we will describe how one can improve the LDL^T slicing algorithm for $\mathcal{H}_\ell$-matrices.

Name	n	t_{dysev} in s (all ev.)	10 Eigenvalues abs. error	10 Eigenvalues t in s	All Eigenvalues abs. error	All Eigenvalues t in s
trid1	64	<0.01	3.5232 e−09	<0.01	3.2186 e−09	0.03
trid2	128	<0.01	3.4594 e−09	0.01	1.3409 e−09	0.06
trid3	256	0.01	3.2238 e−09	0.01	2.0692 e−09	0.16
trid4	512	0.05	9.3048 e−10	0.02	4.2384 e−12	0.37
trid5	1 024	0.36	3.0809 e−09	0.04	1.6036 e−11	0.80
trid6	2 048	2.94	1.3323 e−15	0.07	3.8816 e−12	3.31
trid7	4 096	23.48	1.3323 e−15	0.13	5.9723 e−12	13.51
trid8	8 192	187.89	1.3323 e−15	0.29	1.8445 e−11	58.68
trid9	16 384	1 572.58	2.9310 e−14	0.58	1.2835 e−11	241.38
trid10	32 768	13 053.36	1.3323 e−15	1.27	1.6486 e−11	1 050.28
trid11	65 536	—	—	2.64	—	4 374.16
trid12	131 072	—	—	5.59	—	—
trid13	262 144	—	—	11.40	—	—
trid14	524 288	—	—	23.77	—	—
trid15	1 048 576	—	—	49.15	—	—

Table 5.3: Comparison of errors and computation times for tridiagonal matrices ($[-1, 2, -1]$) computing only 10 eigenvalues ($n/4+5, \ldots, n/4+14$) and all eigenvalues (LAPACK-function `dsyev` is not optimized for tridiagonal matrices).

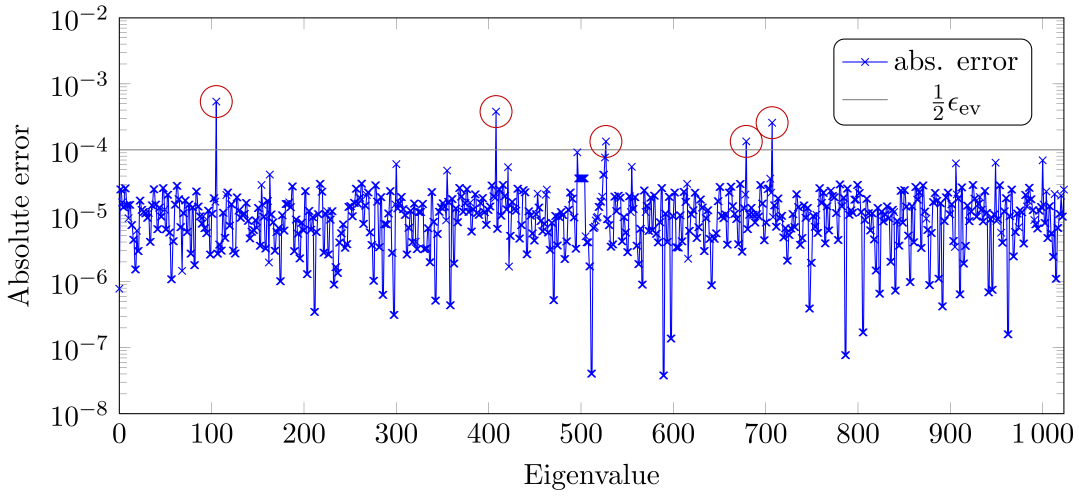


Figure 5.5: Absolute error $|\lambda_i - \hat{\lambda}_i|$ for FEM32 matrix, $\epsilon_{\text{ev}} = 2\,\text{e}{-4}$.

Name	n	t_{dsyev} in s (all ev.)	10 Eigenvalues abs. error	t in s	All Eigenvalues abs. error	t in s
HSS1 r1	64	<0.01	3.2660 e−09	<0.01	1.7203 e−09	0.06
HSS2 r1	128	<0.01	4.0212 e−09	0.02	2.4553 e−09	0.33
HSS3 r1	256	0.01	3.2320 e−09	0.07	2.7721 e−09	1.72
HSS4 r1	512	0.10	4.5453 e−09	0.17	2.6323 e−09	8.39
HSS5 r1	1 024	0.65	3.5624 e−09	0.39	3.1354 e−09	39.76
HSS6 r1	2 048	5.05	4.5539 e−09	0.90	2.2252 e−09	173.37
HSS7 r1	4 096	40.45	4.5020 e−09	2.11	2.2500 e−09	790.44
HSS8 r1	8 192	326.92	4.5912 e−09	4.46	2.8584 e−09	3 368.12
HSS9 r1	16 384	2 700.80	4.5953 e−09	10.29	2.8121 e−09	14 439.43
HSS10 r1	32 768	22 628.85	4.5770 e−09	20.35	2.7312 e−09	55 781.07
HSS11 r1	65 536	—	—	38.87	—	223 925.40
HSS12 r1	131 072	—	—	83.60	—	—
HSS13 r1	262 144	—	—	186.59	—	—
HSS14 r1	524 288	—	—	333.29	—	—
HSS15 r1	1 048 576	—	—	684.15	—	—
HSS5 r1	1 024	0.65	3.1354 e−09	0.38	3.5624 e−09	39.81
HSS5 r2	1 024	0.74	2.5957 e−09	0.54	4.1746 e−09	53.18
HSS5 r3	1 024	0.69	2.7597 e−09	0.73	4.5259 e−09	69.62
HSS5 r4	1 024	0.67	3.1219 e−09	0.86	4.0331 e−09	84.14
HSS5 r8	1 024	0.71	2.4067 e−09	1.74	4.4214 e−09	163.87
HSS5 r16	1 024	0.76	2.3409 e−09	3.46	4.4460 e−09	411.35

Table 5.4: Comparison of errors and computation times for HSS matrices computing only 10 eigenvalues ($n/4+5, \dots, n/4+14$) and all eigenvalues.

Name	n	10 Eigenvalues abs. error	t in s	All Eigenvalues abs. error	t in s
log64	64	2.8210 e−07	0.02	1.3543 e−02	0.14
log128	128	1.2903 e−03	0.11	1.2393 e−02	1.40
log256	256	7.0077 e−03	0.37	8.1165 e−03	8.85
log512	512	3.5651 e−03	1.04	4.1448 e−03	43.78
log1024	1 024	1.7723 e−03	1.97	1.9622 e−03	185.68
log2048	2 048	7.4092 e−04	4.03	9.7573 e−04	711.25
log4096	4 096	6.6052 e−05	6.56	4.8654 e−04	2 469.25
log8192	8 192	5.3191 e−05	12.63	4.2350 e−04	8 774.92
log16384	16 384	6.0408 e−06	20.69	8.0925 e−05	22 165.30

Table 5.5: Comparison of errors and computation times for matrices based on logarithmic kernel computing only 10 eigenvalues ($n/4+5, \dots, n/4+14$) and all eigenvalues.

5.4.1 LDL^T Slicing Algorithm for HSS Matrices

In [109], Xia and Chandrasekaran et al. present an algorithm to compute the Cholesky factorization of a symmetric, positive definite hierarchical semi-separable matrix. Their ideas can be used to construct a similar LDL^T factorization for HSS matrices. Such an algorithm would use the structure of HSS matrices much better, so that the complexity of the LDL^T factorization would be reduced to $\mathcal{O}(k^2n)$.

In Algorithm 5.3 (p. 108) we take [109, Algorithm 2] and apply some small changes to allow the algorithm to compute the LDL^T factorization. The comments are the corresponding lines from the original algorithm. This topic may be investigated further in the future.

5.4.2 LDL^T Slicing Algorithm for $\mathcal{H}$-Matrices

Does Algorithm 5.1 work for $\mathcal{H}$-matrices, too? The answer is yes if we have $\mathcal{O}(n^2)$ storage and $\mathcal{O}(n^3)$ time. The answer is no if linear-polylogarithmic complexity per eigenvalue has to be reached.

We have to use the $\mathcal{H}$-LDL^T factorization for $\mathcal{H}$-matrices. Two additional problems appear: First, in general there is no exact $\mathcal{H}$-LDL^T factorization like in the $\mathcal{H}_\ell$ case. So truncation is required to keep the block-wise ranks at a reasonable size. But then the results are much less accurate and so we encounter wrong decisions, resulting in an incorrect $\nu(\mu)$, which leads to intervals that do not contain the searched eigenvalue, see Figure 5.5. Second, if we use fixed accuracy $\mathcal{H}$-arithmetic we get admissible blocks of large rank for some shifts. This will increase the computational complexity as well as the storage complexity.

We have done some example computations to show the rank growth for different shifts. Therefore, we use the FEM example series from FEM8 to FEM512, see Subsection 2.4.1. Table 5.6 shows the maximal block-wise rank of the factors after the LDL^T factorization of FEMX matrices for different shifts. If the shifted matrix is positive definite, the ranks will stay small. For shifts that are near, for example, the eigenvalue 4 we get large ranks for large matrices. We observe that the maximal block-wise rank is doubled from one column to the next. This shows that Lemma 5.2.2 does not hold for general $\mathcal{H}$-matrices since the rank grows faster than ℓk for large matrices. It follows that the complexity is not in

$$\mathcal{O}\left(k^2 n \left(\log_2 n\right)^4\right),$$

and so for large matrices the computation time grows faster than the expected costs N_i

$$N_i = |EV|\, C_{sp} C_{id} n_i (\log_2 n_i)^4,$$

as we see in Table 5.7. Still, if only a few eigenvalues are required, the computation time is much better than using LAPACK and we can also solve eigenvalue problems that would be otherwise insolvable.

Shift	FEM8	FEM16	FEM32	FEM64	FEM128	FEM256	FEM512
0	8	10	11	11	11	11	11
2	8	11	13	21	37	69	101
4.1	8	11	16	32	61	126	173
4.01	8	11	16	32	64	127	180
4.001	8	11	16	32	64	128	190
4.0001	8	11	16	32	64	128	183
ℓ	1	3	5	7	9	11	13
n	64	256	1 024	4 096	16 384	65 536	262 144

Table 5.6: Maximal block-wise rank after LDL^T factorization for different shifts and different FEM matrices ($\epsilon = 10^{-5}$, block-wise rank 8 before LDL^T factorization, normalized with $\lambda_{\max} = 8$, multiple eigenvalue at 4).

Name	n	t_{dysev} in s	rel. error	t in s	$\frac{t_i}{t_{i-1}}$	$\frac{N_i}{N_{i-1}}$
FEM8	64	<0.01	5.2265 e−07	0.01		
FEM16	256	0.01	2.4938 e−07	0.18	18.00	189.63
FEM32	1 024	0.67	5.7118 e−08	2.91	16.17	42.32
FEM64	4 096	41.87	1.0784 e−06	27.82	9.56	11.61
FEM128	16 384	2 819.50	1.5215 e−05	238.53	6.22	7.41
FEM256	65 536	—	6.3208 e−05	2 151.54	*9.02*	6.82
FEM512	262 144	—	3.2221 e−06	18 699.01	*8.69*	8.24
FEM1024	1 048 576	—	1.2545 e−05	180 456.78	9.65	13.29

Table 5.7: Example of finding the 10 eigenvalues $n/4 + 5, \ldots, n/4 + 14$ of FEMX, ϵ =1 e−5, ϵ_{ev} =1 e−3; *italic entries* are larger than expected.

Name	n	$t_{1\text{ core}}$ in s	$t_{2\text{ core}}$ in s	$t_{1\text{c}}/t_{2\text{c}}$	$t_{4\text{ core}}$ in s	$t_{1\text{c}}/t_{4\text{c}}$	$t_{1\text{c}}/t_{8\text{c}}$
H2 r1	128	0.33	0.18	1.83	0.10	3.30	5.50
H4 r1	512	9.44	4.87	1.94	2.57	3.67	6.60
H6 r1	2 048	219.28	114.69	1.91	60.27	3.64	6.47
H8 r1	8 192	4 022.80	2 151.55	1.87	1 170.63	3.44	5.95
H10 r1	32 768	49 012.24	25 375.91	1.93	15 402.22	3.18	4.90

Table 5.8: Parallelization speedup for different matrices, OpenMP parallelization.

5.4.3 Parallelization

If we search for more than one eigenvalue, we can use some of the computed $\nu(\mu)$ again. For example, since a_i and b_i are the same for all i, at the beginning the first μ_i will be the same, too. After the first LDL^T factorization and the computation of $\nu(M - \mu I) = \nu$, we have two intervals. The interval $[a, \mu]$ contains ν eigenvalues and the interval $[\mu, b]$ contains $n - \nu$ eigenvalues. The computations on the two intervals are independent. We can use two cores, one for each interval, so that we can continue the computations independently. If we have more cores we can increase the number of working cores on the next level to four and so on.

Since there is almost no communication, this will lead to a very good parallelization. All we need is enough storage for the two $\mathcal{H}_\ell$-matrices, M and L, on each core. Since only linear-polylogarithmic storage is required, this should often be the case.

In the first attempt we have used OpenMP [84] to parallelize the program code used for the numerical examples. This leads to a simple parallelization that should be possible to improve. Table 5.8 shows the timing results on just one node of Otto, now we use up to eight cores. The speedup from one core to four cores is about 3.3. For eight cores the increase is around 5. This is already a satisfying value compared with the parallelization of other algorithms, like the parallelization of the LAPACK-function `dlahqr` (QR algorithm for unsymmetric eigenvalue problems) shows a speedup of 2.5 on four nodes [62].

Further, we have a parallelization using Open MPI [97] and a master-slave structure. The master distributes the work, by giving each slave an interval. The slave computes all eigenvalues in this interval if there are not more than $2m$ eigenvalues in the interval. If there are more eigenvalues then the slave computes only the inner m eigenvalues. The slave first computes the smallest and the largest eigenvalue of this set, and tells the master that he will not compute the other eigenvalues. So the master is quickly notified that there are two intervals, where no slave is computing eigenvalues. The master manages a list of all these free intervals and gives each idle slave a new interval immediately.

This parallelization is more sophisticated and leads to higher speed-ups. In addition, Open MPI permits the usage of different nodes of a cluster so we are no longer restricted to one node as in the OpenMP parallelization. Tests of this implementation on our cluster Otto show a very good scalability of the algorithm up to 288 parallel processes (24 nodes with each 12 cores). The speed-up was 230.50 for 287 slave processes and one master, see Table 5.9. The efficiency of the parallelization for 288 is still over 80%.

We have seen that the LDL^T slicing algorithm is very well parallelizable, since the computations for disjoint intervals do not influence each other.

No. of Processes	t in s	Speedup	Efficiency
1	16 564.58	1.00	1.00
2+1	8 340.22	1.99	0.66
4+1	4 044.13	4.10	0.82
6+1	2 678.95	6.18	0.88
11+1	1 494.33	11.08	0.92
23+1	713.80	23.21	0.96
35+1	476.44	34.77	0.96
47+1	364.30	45.47	0.95
95+1	188.92	87.68	0.91
191+1	100.61	164.64	0.86
287+1	71.86	230.50	0.80
383+1	61.91	267.56	0.70

Table 5.9: Parallelization speedup of the Open MPI parallelization for H9 r1.

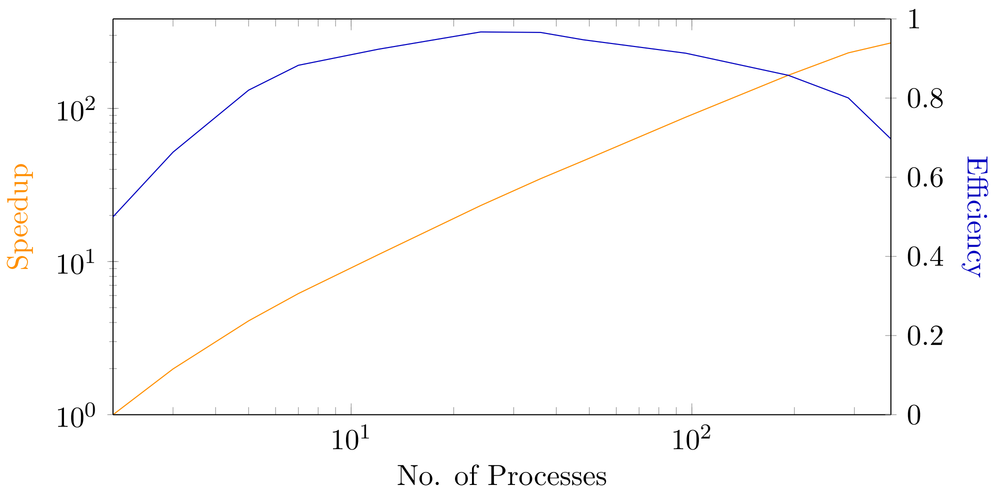


Figure 5.6: Parallelization speedup and efficiency of the Open MPI parallelization for H9 r1.

5.4.4 Eigenvectors

Often, the eigenvectors of some eigenvalues are also of interest. The LDL^T slicing algorithm does not compute the eigenvectors. The last LDL^T factorization can be used as a preconditioner for a shifted preconditioned inverse iteration, see Chapter 6. Since we have a good approximation of the eigenvalue, we expect fast convergence.

If the eigenvectors are clustered we can compute the corresponding invariant subspace by a subspace version of preconditioned inverse iteration.

As well as computing the eigenvector, this will give us an improved approximation to the eigenvalue. This may be used to detect and remedy wrong decisions in the case of approximative arithmetic.

5.5 Conclusions

We have discussed the application of the old and nearly forgotten slicing-the-spectrum algorithm for computing selected eigenvalues of symmetric matrices to the class of $\mathcal{H}_\ell$-matrices. The LDL^T slicing algorithm uses the special structure of symmetric $\mathcal{H}_\ell$-matrices, which makes the repeated computation of LDL^T factorizations (which is the obstacle to its use for general dense matrices) a feasible computational task. In particular, the LDL^T slicing algorithm enables us to compute an interior eigenvalue of a symmetric $\mathcal{H}_\ell$-matrix in linear-polylogarithmic complexity. Numerical results confirm this. For the computation of a single or a few interior eigenvalues the algorithm is superior to existing ones. It is less efficient for computing all eigenvalues of symmetric $\mathcal{H}_\ell$-matrices, but due to the efficient use of memory, it allows to solve much larger dense eigenvalue problems within the considered class of matrices than simply applying the methods available in LAPACK.

We may also use the LDL^T slicing algorithm for computing the eigenvalues of general, symmetric $\mathcal{H}$-matrices. But then the algorithm is no longer of linear-polylogarithmic complexity. Nevertheless, again the computation of a few interior eigenvalues is possible for problem sizes that by far exceed the capabilities of standard Numerical Linear Algebra algorithms for symmetric matrices.

For special classes within the set of $\mathcal{H}_\ell$-matrices, like hierarchical semi-separable (HSS) matrices, there might be even more efficient variants if the special structure is exploited in the LDL^T factorization.

Finally we have seen that multi-core architectures can be exploited easily and lead to fairly good speed-up and parallel efficiency.

Since we are still missing an eigenvalue algorithm for general $\mathcal{H}$-matrices, we will investigate an algorithm based on preconditioned inverse iteration in the next chapter. Further this is the only algorithm that computes the eigenvectors, too.

Algorithm 5.3: Generalized LDL^T factorization based on a generalized Cholesky factorization for HSS matrices, see Algorithm 2 [109].

```
   Input: HSS matrix $H$ with $n$ nodes in the HSS tree, $H = H^T$
   Output: HSS matrix containing $L$ and $D$, with $H = LDL^T$
1  Allocate space for a stack.
2  for each node $i = 1, \dots, n-1$ do
3      if $i$ is a non-leaf node then
4          Pop four matrices $\tilde{E}_{c_2}, \tilde{U}_{c_2}, \tilde{E}_{c_1}, \tilde{U}_{c_1}$ from the stack, where $c_1$, $c_2$ are the
           children of $i$.
5          Obtain $E_i$ and $U_i$ by
               $$E_i = \begin{bmatrix} \tilde{E}_{c_1} & \tilde{U}_{c_1} B_{c_1} \tilde{U}_{c_2}^T \\ \tilde{U}_{c_2} B_{c_1} \tilde{U}_{c_1}^T & \tilde{E}_{c_2} \end{bmatrix}, U_i = \begin{bmatrix} \tilde{U}_{c_1} R_{c_1} \\ \tilde{U}_{c_2} R_{c_2} \end{bmatrix}.$$
6      end
7      Compress $U_i$ by the QL factorization $\hat{U}_i = Q_i U_i = \begin{bmatrix} 0 \\ \tilde{U}_i \end{bmatrix}$ and push $\tilde{U}_i$ onto the
       stack
8      Update $E_i$ with $\tilde{E}_i = Q_i^T E_i Q_i$. Factorize $\tilde{E}_i$ with
           $$\hat{E}_i = \begin{bmatrix} L_i & 0 \\ E_{i;2,1} L_i^{-T} D_i^{-1} & I \end{bmatrix} \begin{bmatrix} D_i & 0 \\ 0 & \tilde{E}_i \end{bmatrix} \begin{bmatrix} L_i^T & D_i^{-1} L_i^{-1} E_{i;1,2} \\ 0 & I \end{bmatrix},$$
       and obtain the Schur complement $\tilde{E}_i$ as
           $$\tilde{E}_i = E_{i;2,2} - E_{i;2,1} L_i^{-T} D_i^{-1} L_i^{-1} E_{i;1,2}.$$
       Push $\tilde{E}_i$ onto the stack.
       If $D_i$ is singular, stop the algorithm and return error message $0 \in \Lambda(H)$.
       /* Update $E_i$ with $\tilde{E}_i = Q_i^T E_i Q_i$.  Factorize $\tilde{E}_i$ with
           $$\hat{E}_i = \begin{bmatrix} L_i & 0 \\ E_{i;2,1} L_i^{-T} & I \end{bmatrix} \begin{bmatrix} L_i^T & L_i^{-1} E_{i;1,2} \\ 0 & \tilde{E}_i \end{bmatrix},$$
           and obtain the Schur complement $\tilde{E}_i$ as
           $$\tilde{E}_i = E_{i;2,2} - E_{i;2,1} L_i^{-T} L_i^{-1} E_{i;1,2}.$$
           Push $\tilde{E}_i$ onto the stack.                                    */
9      If $D_i = 0$, then stop and note that 0 is an eigenvalue of $H$.
10 end
11 For root $n$, compute the Cholesky factorization $E_n = L_n D_n L_n^T$
   /* For root $n$, compute the Cholesky factorization $E_n = L_n L_n^T$        */
```

CHAPTER

SIX

COMPUTING EIGENVALUES BY VECTOR ITERATIONS

Contents

In the last chapters we have seen two eigenvalue algorithms that work efficently only for a small subset of hierarchical matrices. The extension of these algorithms for the computation of all eigenvalues to general $\mathcal{H}$-matrices fails due to too large block-wise ranks. In this chapter we will investigate algorithms for the computation of one or some few eigenvalues. These algorithms are all based on the power iteration, which will be investigated first.

6.1 Power Iteration

The *power iteration*, or sometimes called power method, see [107],

$$x_{i+1} = Mx_i / \left\|Mx_i\right\|_2 ,$$

is one of the simplest methods for the computation of the eigenvector corresponding to the (absolute) largest eigenvalue. An approximation to the spectral norm $\|M\|_2$ is given by the divisor $\|Mx_i\|_2$. If M is symmetric positive definite, then

$$\|M\|_2 = \lambda_n \geq \lambda_i \quad \forall i = \{1, \dots, n-1\}.$$

The main advantage of the power iteration is that the simple iteration formula only requires matrix-vector products. This makes the power iteration the method of choice for eigenvalue problems, where the matrix M is only available through matrix-vector products, e.g., in the PageRank problem [86]. Since the previous iterates are not required, it is sufficient to store one vector x. This is an advantage for large n, since sometimes it is impossible to store a full Krylov subspace due to memory limitation.

The disadvantage of the power iteration is the slow convergence, especially if the quotient λ_n/λ_{n-1} is near 1. Furthermore, in exact arithmetic the power iteration may converge to the second largest eigenvector if the starting vector x_0 is orthogonal to the eigenvector v_n corresponding to the largest eigenvalue. If x_0 is orthogonal to more eigenvectors the power iteration may even converge to another eigenvector. This problem does not occur in approximate arithmetic, though it slows down the convergence if x_0 is orthogonal to v_n.

If the eigenspace to λ_n is of a dimension larger than one, then the power iteration computes an arbitrary eigenvector out of this subspace. If the whole subspace is needed, then one should use a subspace version of the power iteration. The subspace version is called simultaneous iteration.

There are different possibilities to compute the second largest eigenvalue. One way is to compute the eigenvector v_n and then restart the process, but orthogonalize all x_i against v_n. This deflation procedure guarantees to find v_{n-1} (if the starting vector x_0 is not orthogonal to v_{n-1}). The other way is to use a subspace version of the power iteration:

$$\begin{aligned} X_{i+1} &= MX_i \\ X_{i+1} &= \frac{X_{i+1}}{\|X_{i+1}\|_2}, \end{aligned}$$

where $X_i \in \mathbb{R}^{n\times d}$. This will finally give an invariant subspace corresponding to the d largest eigenvalues if $\lambda_{n-d+1} > \lambda_{n-d}$. The subspace version has the disadvantage that in later steps it computes some matrix-vector products for already converged vectors. These products are unnecessary and increase the costs but, on the other hand, the deflation is not for free, too.

6.1.1 Power Iteration for Hierarchical Matrices

The power iteration requires only one arithmetic operation, the matrix-vector product. In the $\mathcal{H}$-arithmetic there is a cheap and efficient $\mathcal{H}$-matrix-vector product of almost linear complexity, as shown in Lemma 2.3.10.

The power iteration, as well as the simultaneous iteration, was used for the computation of eigenpairs of hierarchical matrices. In, e.g., [48, 21, 50, 20] the power iteration was used to compute the spectral norm $\|M\|_2$ of hierarchical matrices. The spectral norm of $\mathcal{H}^2$-matrices is computed in [16, 18] by the power iteration. Lintner [74] uses a simultaneous iteration together with a Schur-Rayleigh-Ritz step for the computation of eigenvalues of the generalized eigenvalue problem

$$Ax = \mu M x.$$

6.1.2 Inverse Iteration

The *inverse iteration* is related to the power iteration, with the only difference being that one substitutes the matrix M by M^{-1}. The largest eigenvalue of M^{-1} is the smallest in magnitude eigenvalue of M, which is computed by the inverse iteration

$$x_{i+1} = \frac{(M-\mu I)^{-1} x_i}{\left\|(M-\mu I)^{-1} x_i\right\|}, \quad \mu = 0.$$

By incorporating shifts $\mu \neq 0$, the convergence properties of the inverse iteration can be improved. One can also use shifts $\mu \neq 0$ to compute other eigenvalues than the smallest in magnitude. The convergence rate is determined by the quotient

$$\frac{\lambda_j - \mu}{\lambda_k - \mu},$$

with λ_j being the nearest eigenvalue to μ and λ_k being the second nearest.

The inverse iteration is often regarded as an invention of Wielandt, see [66] for the history of inverse iteration.

The $\mathcal{H}$-matrix inverse $M_{\mathcal{H}}^{-1}$ is in general not exact enough for using the largest eigenvalue of $M_{\mathcal{H}}^{-1}$ as approximation to the smallest eigenvalue of M. In the next section we will discuss a possibility to overcome this problem by applying $M_{\mathcal{H}}^{-1}$ only to the residual of the eigenvalue problem.

6.2 Preconditioned Inverse Iteration for Hierarchical Matrices

The preconditioned inverse iteration [80] is an efficient method to compute the smallest eigenpair of a symmetric positive definite matrix M. Here we use this method to find the smallest eigenvalues of an $\mathcal{H}$-matrix. For the preconditioning, we use the approximate inverse of M resp. the approximate Cholesky decomposition of M. Both can be computed in linear-polylogarithmic complexity, so that the computation of one eigenvalue is cheap.

We extend the ideas to the computation of inner eigenvalues by computing an invariant subspaces S of $(M-\mu I)^2$ by subspace preconditioned inverse iteration. The eigenvalues of the generalized matrix Rayleigh quotient $\mu_M(S)$ are the desired inner eigenvalues of M. The idea of using $(M-\mu I)^2$ instead of M is known as the folded spectrum method [104].

The preconditioned inverse iteration (PINVIT) is a so called matrix-free method, not requiring to have M or M^{-1} available as matrices, but as functions $x \to Mx$ and $x \to M^{-1}x$. PINVIT computes the smallest eigenvalue and the corresponding eigenvector by minimizing the Rayleigh quotients.

Definition 6.2.1: (Rayleigh quotient)
The function

$$\mu(x,M) : \mathbb{R}^n \setminus \{0\} \times \mathbb{R}^{n\times n} \to \mathbb{R} : (x,M) \to \mu(x) = \mu(x,M) = \frac{x^T M x}{x^T x} \tag{6.1}$$

is called the *Rayleigh quotient.*

Corollary 6.2.2: The Rayleigh quotient of the eigenvector v_i is $\mu(v_i) = \lambda_i$.

Proof.

$$\mu(v_i, M) = \frac{v_i^T M v_i}{v_i^T v_i} = \frac{v_i^T \lambda_i v_i}{v_i^T v_i} = \lambda_i.$$

■

The definition of the Rayleigh quotient can be generalized to full rank rectangular matrices X.

Definition 6.2.3: (matrix Rayleigh quotient, generalized scalar Rayleigh quotient) [1]
Let X be an element of the Grassmann manifold

$$X \in Gr(d,n) = \left\{ X \in \mathbb{R}^{n\times d}, \text{column rank}\, X = d \right\},$$

$d \leq n$. The *matrix Rayleigh quotient* is defined by

$$R(X,M) = \left(X^T X\right)^{-1} X^T M X. \tag{6.2}$$

The function

$$\mu(X,M) : \mathbb{R}^{n\times d} \times \mathbb{R}^{n\times n} \to \mathbb{R} : (X,M) \to \mu(X,M) = \operatorname{tr}\left(R(X,M)\right) \tag{6.3}$$

is called the *generalized scalar Rayleigh quotient.*

Let S span an invariant subspace with $MS = SY$. The matrix Rayleigh quotient of S is

$$R(S, M) = \left(S^T S\right)^{-1} S^T M S = \left(S^T S\right)^{-1} \left(S^T S\right) Y = Y.$$

This section is structured as follows. In the next subsection we give a brief summary of the preconditioned inverse iteration. The second subsection contains a short review of hierarchical matrices and their approximate inverses and Cholesky decompositions. Both concepts are combined in the third subsection to a preconditioned inverse iteration for hierarchical matrices. Afterwards, we use the ideas of the folded spectrum method [104] to extend the preconditioned inverse iteration to the computation of interior eigenvalues. Then numerical results are presented substantiating the convergence properties and showing that, for non-sparse matrices, the computation of the eigenvalues is superior to existing algorithms. We finally present some concluding remarks.

6.2.1 Preconditioned Inverse Iteration

The first iterative eigensolvers with preconditioner date back to the late 1950s, see [68] or [82] for references. In [82] Neymeyr classifies the different schemes of preconditioned inverse iteration and proves some statements on their convergence rates. This can also be found in [80, 81, 70]. Further, he gives some motivation for using preconditioned inverse iteration and so we will pick the following one from there:

The preconditioned inverse iteration, PINVIT for short, can be regarded as a gradient-based minimization method minimizing the Rayleigh quotient. We have

$$\mu(x) = \frac{x^T M x}{x^T x}$$

and

$$\nabla \mu(x) = \frac{2}{x^T x} \left(Mx - x\mu(x)\right).$$

This leads to the gradient method

$$x_i := x_{i-1} - \alpha \left(M x_{i-1} - x_{i-1}\mu(x_{i-1})\right).$$

The convergence of this method is too slow, so one should use the acceleration by preconditioning the residual $r = Mx - x\mu(x)$ with the preconditioner T^{-1}:

$$\nabla_T \mu(x) = \frac{2}{x^T x} T^{-1} \left(Mx - x\mu(x)\right).$$

We finally get the update equation:

$$x_i := x_{i-1} - T^{-1} r_{i-1} = x_{i-1} - T^{-1} \left(M x_{i-1} - x_{i-1}\mu(x_{i-1})\right). \tag{6.4}$$

A second really simple argument for using the PINVIT is that there is a nice sharp bound on the convergence rate of PINVIT, as proved in [80, 81, 70] and recently in a shorter form in [71]:

Theorem 6.2.4: [71, Theorem 1.1]
Let $x \in \mathbb{R}^n$ and $\lambda_j \leq \mu(x) < \lambda_{j+1}$. If the preconditioner T^{-1} satisfies

$$\left\|\mathcal{I} - T^{-1}M\right\|_M \leq c < 1, \tag{6.5}$$

then it is true that for the Rayleigh quotient of the next iterate $\mu(x')$, either $\mu(x') < \lambda_j$ or $\lambda_j \leq \mu(x') < \mu(x)$. In the latter case,

$$\frac{\mu(x') - \lambda_j}{\lambda_{j+1} - \mu(x')} \leq \gamma^2 \frac{\mu(x) - \lambda_j}{\lambda_{j+1} - \mu(x)},$$

where

$$\gamma = 1 - (1-c)\left(1 - \frac{\lambda_j}{\lambda_{j+1}}\right) \tag{6.6}$$

is the convergence factor.

There are some variations of this method that slightly improve the convergence. One can use the optimal vector in the subspace spanned by x_{i-1} and $T^{-1}r_{i-1}$ as the next iterate x_i. Neymeyr calls this variation PINVIT(2), although it is also called preconditioned steepest descent. Further, one can use the subspace spanned by x_{i-1}, x_{i-2} and $T^{-1}r_{i-1}$. This method is called PINVIT(3) in Neymeyr's classification. In [69] Knyazev changed the subspace into the more stable one $\{x_{i-1}, p_{i-1}, T^{-1}r_{i-1}\}$, the resulting method is called linear optimal (block) preconditioned conjugate gradient method (LOBPCG)

$$\begin{aligned}
p_0 &= 0, \\
w_i = T^{-1}r_{i-1} &= T^{-1}(Mx_{i-1} - x_{i-1}\mu_{i-1}), \\
x_i &= w_i + \tau_{i-1}x_{i-1} + \gamma_{i-1}p_{i-1}, \\
p_i &= w_i + \gamma_{i-1}p_{i-1}.
\end{aligned}$$

The method is called PINVIT(q, s) when one replaces x_i by the subspace X_i of dimension s.

The convergence factor γ in the subspace case depends, among other factors, on the quotient

$$\frac{\lambda_s}{\lambda_{s+1}}. \tag{6.7}$$

If this quotient is one, then the subspace is not unique.

We will compute the eigenvalues of a hierarchical matrix using preconditioned inverse iteration in the versions PINVIT(q, s), $q \leq 3$. To do this we need an approximative $\mathcal{H}$-inversion resp. $\mathcal{H}$-Cholesky decomposition. In the next subsection we review special algorithms, ensuring that Condition 6.5 is fulfilled.

Algorithm 6.1: Adaptive Inversion of an $\mathcal{H}$-Matrix [48].

```
    Input: $M \in \mathcal{H}(T_{I\times I})$, $\tilde{c} \in \mathbb{R}$
    Output: $M^{-1}_{\text{adap.},\mathcal{H}}$, with $\left\|\mathcal{I} - M^{-1}_{\text{adap.},\mathcal{H}} M\right\|_2 < \tilde{c}$
1   Compute $C_{sp}(M)$ and $\|M\|_2^{\mathcal{H}}$;
2   Compute $M^{-1}_{\mathcal{H}}$ with $\epsilon_{\text{local}} = \tilde{c}/(C_{sp}(M)\,\|M\|_2^{\mathcal{H}})$;
3   $\delta^0_M := \left\|\mathcal{I} - M^{-1}_{\mathcal{H}} M\right\|_2^{\mathcal{H}} / \tilde{c}$;
4   while $\delta^i_M > 1$ do
5       Compute $M^{-1}_{\text{adap.},\mathcal{H}}$ with $\epsilon_{\text{local}} := \epsilon_{\text{local}}/\delta^i_M$ or $\epsilon_{\text{local}} := \epsilon_{\text{local}}/\max_i \delta^i_M$;
6       $\delta^i_M := \left\|\mathcal{I} - M^{-1}_{\mathcal{H}} M\right\|_2^{\mathcal{H}} / \tilde{c}$;
7   end
```

6.2.2 The Approximate Inverse of a Hierarchical Matrix

In the next section we use the $\mathcal{H}$-inverse as preconditioner in the PINVIT for $\mathcal{H}$-matrices. As such, here we review the adaptive $\mathcal{H}$-inverse from [48], see Algorithm 6.1.

The preferred method for the inversion is the recursive block Gaussian elimination. The block-wise computations of the recursive block Gaussian elimination are done with the $\mathcal{H}$-accuracy ϵ. Unfortunately, this does not guarantee accuracy of the $\mathcal{H}$-inverse $M^{-1}_{\mathcal{H}}$. Grasedyck designed Algorithm 6.1 [48] to solve the problem by estimating the $\mathcal{H}$-accuracy ϵ required to get an approximate inverse $M^{-1}_{\text{adap.},\mathcal{H}}$ satisfying

$$\left\|\mathcal{I} - M^{-1}_{\text{adap.},\mathcal{H}} M\right\|_2^{\mathcal{H}} < \tilde{c}$$

for a prescribed tolerance $\tilde{c}$. If $LL^T = M$ is the Cholesky decomposition of M, then Equation (6.4) becomes

$$\left\|I - T^{-1}M\right\|_M = \left\|L\left(I - T^{-1}M\right)\right\|_2 \leq \|L\|_2 \left\|I - T^{-1}M\right\|_2 = \sqrt{\|M\|_2}\left\|I - T^{-1}M\right\|_2 .$$

Thus, we have to choose

$$\tilde{c} = \frac{c}{\sqrt{\|M\|_2}} \text{ and } c < 1,$$

to get an inverse $M^{-1}_{\text{adap.},\mathcal{H}}$ that fulfills Condition (6.5), so that we can use $T^{-1} := M^{-1}_{\text{adap.},\mathcal{H}}$ as preconditioner.

The complexity of this process and the rank of the result is in general unknown. There are only some results for special matrices. In [57] it was shown that the inverse of FEM matrices of uniformly elliptic operators with L^∞-coefficients can be approximated by an $\mathcal{H}$-matrix if the triangularization is regular. In [17] Börm investigated the problem with a new approach using Clément interpolations instead of Green's functions and L^2-orthogonal projectors. Essentially it is shown in [17] that there are good $\mathcal{H}$-approximations to the inverses of elliptic partial differential operators on meshes of sufficient regularity. Further, in [17, Theorem 3] it is shown that the accuracy improves exponentially while the ranks grow like a polynomial p of order $d+1$, with d as the dimension of the partial differential operator. For quasi-uniform meshes p can be chosen like $p \sim c + \log_2 n$ and one gets block-wise ranks $k \sim C_{dim}(c + \log_2 n)^{d+1}$, where C_{dim} is a constant depending on the discretization [17].

The evaluation costs of the preconditioner $M^{-1}_{\text{adap.},\mathcal{H}}$, computed by Algorithm 6.1, depend on the ranks of the admissible blocks. These ranks depend on the one hand on the used $\mathcal{H}$-accuracy ϵ and, as such, on c also. On the other hand, the ranks depend on the properties of M, especially on the maximal block rank k and the norms $\|M\|_2$ and $\|M^{-1}\|_2$. To have both is quite natural as a better preconditioner is more expensive and a preconditioner for a matrix of better condition is cheaper.

The method of how to find the optimal spectral equivalent preconditioner in the $\mathcal{H}$-format with the minimal block-wise rank to a given (symmetric positive definite) $\mathcal{H}$-matrix M is still an open problem [56].

Experiments show that the $\mathcal{H}$-inversion is expensive, though it is of linear-polylogarithmic complexity. One alternative is the $\mathcal{H}$-Cholesky decomposition, which we investigate in the next subsection. In Section 6.2.6 we will see that the adaptive Cholesky decomposition is much cheaper than the adaptive $\mathcal{H}$-inversion.

6.2.3 The Approximate Cholesky Decomposition of a Hierarchical Matrix

For the preconditioned inverse iteration we need a preconditioner for the symmetric matrix M. The Cholesky decomposition, together with a solver for upper and lower triangular systems of equations gives such a preconditioner, too.

The Cholesky decomposition of hierarchical matrices is computed by a block recursive algorithm, see Algorithm 2.2 or [73, 56].

The $\mathcal{H}$-Cholesky decomposition computes a preconditioner of a certain accuracy $\tilde{c}$ measured by

$$\left\|I - L^{-1}L^{-T}M\right\|_2 \leq \tilde{c},$$

where L^{-1} and L^{-T} are the operators for the forward resp. backward solution process of the matrix equation $LX = M$. As for the $\mathcal{H}$-inversion, the accuracy $\tilde{c}$ depends on C_{sp}, $\|M\|$, cond (M) and the $\mathcal{H}$-arithmetic accuracy ϵ. So a priori we do not know how small ϵ has to be to get a preconditioner of the necessary accuracy for the preconditioned inverse iteration.

An adaptive Cholesky decomposition of an $\mathcal{H}$-matrix can be computed in a similar fashion to adaptive $\mathcal{H}$-inversion in Algorithm 6.1. We use the same estimate $\epsilon_{\text{local}} =$

Algorithm 6.2: Adaptive Cholesky Decomposition of an $\mathcal{H}$-Matrix [48].

```
   Input: $M \in \mathcal{H}(T_{I\times I})$, $\tilde{c} \in \mathbb{R}$
   Output: $L_{\text{adap.},\mathcal{H}}$, with $\left\|\mathcal{I} - L_{\text{adap.},\mathcal{H}}^{-T} L_{\text{adap.},\mathcal{H}}^{-1} M\right\|_2 < \tilde{c}$
 1 Compute $C_{sp}(M)$ and $\|M\|_2^{\mathcal{H}}$;
 2 Compute $L_{\mathcal{H}}$ =$\mathcal{H}$-Cholesky-decomposition($M, I$) with
   $\epsilon_{\text{local}} = \tilde{c}/(C_{sp}(M)\,\|M\|_2^{\mathcal{H}})$;
 3 $\delta_0 := \left\|\mathcal{I} - L_{\mathcal{H}}^{-T} L_{\mathcal{H}}^{-1} M\right\|_2^{\mathcal{H}} / \tilde{c}$;
 4 while $\delta_i > 1$ do
 5     $\epsilon_{\text{local}} := \epsilon_{\text{local}}/\delta_i$ or $\epsilon_{\text{local}} := \epsilon_{\text{local}}/\max_i \delta_i$;
 6     Compute $L_{\mathcal{H}}$ =$\mathcal{H}$-Cholesky-decomposition($M, I$) with $\epsilon_{\text{local}}$;
 7     $\delta_i := \left\|\mathcal{I} - L_{\mathcal{H}}^{-T} L_{\mathcal{H}}^{-1} M\right\|_2^{\mathcal{H}} / \tilde{c}$;
 8 end
```

$\tilde{c}/(C_{sp}(M)\,\|M\|_2^{\mathcal{H}})$ for the $\mathcal{H}$-arithmetic accuracy. We compute the $\mathcal{H}$-Cholesky decomposition with ϵ_{local} and compute

$$\eta(\epsilon_{\text{local}}) = \left\|I - L^{-T}L^{-1}M\right\|_2.$$

Further, we assume that the accuracy of the preconditioner η depends linearly on the $\mathcal{H}$-arithmetic accuracy ϵ:

$$\eta(\epsilon_{\text{local}}) = \gamma\epsilon_{\text{local}}.$$

With the pair ϵ_{local} and $\eta(\epsilon_{\text{local}})$ we make a guess for γ. The quotient $\tilde{c}/\gamma$ yields a new estimate for the $\mathcal{H}$-arithmetic accuracy. Unfortunately, the dependency between ϵ and η is not linear, so there is no guarantee that the second $\mathcal{H}$-Cholesky decomposition is exact enough. Maybe we have to repeat the process until the preconditioner fulfills Equation (6.5). All these steps are summarized in Algorithm 6.2.

In the next subsection we apply PINVIT from the previous section to $\mathcal{H}$-matrices using the $\mathcal{H}$-inversion or the $\mathcal{H}$-Cholesky decomposition.

6.2.4 PINVIT for $\mathcal{H}$-Matrices

In this section we will investigate what happens if we use PINVIT to compute the eigenvalues of an $\mathcal{H}$-matrix. For generality we use the subspace version of PINVIT, see [83] for details. We assume that the dimension d of the subspace is small in comparison to n and especially small enough to store rectangular matrices of dimension $n \times d$ in the dense format. We will use Algorithm 6.3, which is slightly different from the one in [80]. In [80] Neymeyr uses the Ritz vectors and Ritz values instead of X and $\mu(X)$.

Algorithm 6.3: Hierarchical Subspace Preconditioned Inverse Iteration.

```
Input: $M \in \mathbb{R}^{n\times n}$, $X_0 \in \mathbb{R}^{n\times d}$ e. g. randomly chosen
Output: $X_p \in \mathbb{R}^{n\times d}$, $\mu \in \mathbb{R}^{d\times d}$, with $\|MX_p - X_p\mu\| \leq \epsilon$
1  Orthogonalize $X_0$;
2  $\mu := X_0^T M X_0$;
3  $R := MX_0 - X_0\mu$;
4  $T^{-1} = (M)_{\mathcal{H}}^{-1}$; or                /* Choose one of the preconditioners. */
5  $L =$ adaptive $\mathcal{H}$-Cholesky decomposition($M$)$\Rightarrow T^{-1}v :=$ Solve $LL^Tx = v$;
6  $i := 1$;
7  while $\left\|T^{-1}R\right\|_F > \epsilon$ do
8      $i := i + 1$;
9      $X_i := X_{i-1} - T^{-1}R$;
10     Orthogonalize $X_i$;
11     $\mu := X_i^T M X_i$;
12     $R := MX_i - X_i\mu$;
13 end
```

The latter would lead to the following steps replacing the computation of the residual in the last line of the for-loop in Algorithm 6.3:

$$\begin{aligned} QDQ^T &:= \texttt{eigendecomposition}(\mu); \\ X_i &:= X_iQ; \\ R &:= MX_i - X_iD. \end{aligned}$$

These changes lead to the following update equation

$$\begin{aligned} X_i &= X_{i-1}Q - (M)_{PC}^{-1}\left(MX_{i-1}Q - X_{i-1}QD\right), \\ &= X_{i-1}Q - (M)_{PC}^{-1}\left(MX_{i-1} - X_{i-1}QQ^T\mu\right)Q, \end{aligned} \tag{6.8}$$

where $(M)_{PC}^{-1}$ has to be substituted by $L^{-T}L^{-1}$ if the Cholesky decomposition is used as preconditioner. Since Q is orthogonal and square, it holds that $QQ^T = \mathcal{I}$. The subspace versions of Equation (6.4) and Equation (6.8) only differ in the factor Q, which is multiplied from the right hand side. This means that the subspaces spanned by X_i are identical. The orthogonal Q causes only an orthogonal transformation of the spanning vectors within the spanned subspaces. Numerical tests do not show an advantage of the diagonalized version. The additional solutions of the small eigenvalue problems produce additional costs, so that Algorithm 6.3 is cheaper.

Algorithm 6.3 is equivalent to the algorithm SPINVIT in [83] or [82], so the convergence analysis in the literature can be used.

Remark 6.2.5: PINVIT can be applied to generalized eigenvalue problems

$$Mx = \lambda Nx,$$

too. Then we have to orthogonalize in lines 1 and 10 with respect to N, so that $X^TNX = \mathcal{I}$. Furthermore, the computation of the residual in lines 3 and 12 has to be changed to

$$R := MX_i - NX_i\mu.$$

The main advantage of $\mathcal{H}$-matrices is the almost linear complexity. The PINVIT for $\mathcal{H}$-matrices benefits from the cheap arithmetic.

The computation of the preconditioner has to be done once and is of linear-polylogarithmic complexity. The non-adaptive $\mathcal{H}$-inversion, as well as the $\mathcal{H}$-Cholesky decomposition, have a complexity of $\mathcal{O}(k^2n\,(\log_2 n)^2)$.

Further, we need one matrix-vector product with M and one evaluation of the preconditioner, one matrix-vector product with $M_{\mathcal{H}}^{-1}$ or two solutions with the triangular L, L^T, per iteration step. Both products have a complexity equal to the storage complexity of M resp. $M_{\mathcal{H}}^{-1}$. The handling of X, R and μ requires some dense arithmetic on $n \times d$ matrices with $\mathcal{O}(d^2n)$ flops. So one iteration has a lower complexity than the $\mathcal{H}$-inversion. Since the number of iterations is independent of the matrix dimension, the dominant computation in the whole process is the inversion of M.

Theorem 6.2.6: *(Convergence Theorem)*
Let $x \in \mathbb{R}^n$ and $\lambda_j < \mu(x) < \lambda_{j+1}$. If $M_{\mathcal{H}}^{-1}$ is computed by Algorithm 6.1 or $L_{\mathcal{H}}$ is computed by Algorithm 6.2 and $\tilde{c} = \frac{c}{\sqrt{\|M\|_2}}$ with $c < 1$, then the Rayleigh quotient of the next iterate x' computed by Algorithm 6.3 either satisfies $\mu(x') < \lambda_j$ or $\lambda_j \leq \mu(x') < \mu(x)$ with

$$\frac{\mu(x') - \lambda_j}{\lambda_{j+1} - \mu(x')} \leq \gamma^2 \frac{\mu(x) - \lambda_j}{\lambda_{j+1} - \mu(x)},$$

where

$$\gamma = 1 - (1-c)\left(1 - \frac{\lambda_j}{\lambda_{j+1}}\right)$$

is the convergence factor.

Proof. In the description of Algorithm 6.1 we show that $M_{\mathcal{H}}^{-1}$ for $\tilde{c} = \frac{c}{\sqrt{\|M\|_2}}$ fulfills

$$\left\|I - M_{\mathcal{H}}^{-1}M\right\|_M < c.$$

So we can apply Theorem 6.2.4.

■

Stopping Criterion

If the subspace spanned by X converges to an invariant subspace, then the residual $\|R\|$ converges to zero. Since we already compute R for the next iterate,

$$\|R\|_F < \epsilon \tag{6.9}$$

is a cheap stopping criterion. In [107, p. 173–174] the following fact is shown for vectors x.

Lemma 6.2.7: [107, p. 173–174]
Let x be a normalized approximation to the eigenvector v_j corresponding to the eigenvalue λ_j and let $\mu = x^T Mx$ be the Rayleigh quotient of x. Further, we need a constant δ, with

$$\delta < |\lambda_i - \mu|, \quad \forall i \in \{1, \dots, n\} \setminus \{j\}.$$

If $\|Mx - \mu x\|_2 = \epsilon < \delta$, then

$$|\mu - \lambda_j| < \frac{\frac{\epsilon^2}{\delta}}{\left(1 - \frac{\epsilon^2}{\delta^2}\right)}. \tag{6.10}$$

Relation (6.10) shows that the Rayleigh quotient has an error of order ϵ^2, if $\delta \gg \epsilon$.

In this section we have described the application of PINVIT to hierarchical matrices. This leads to an algorithm of almost linear complexity for the computation of the smallest eigenvalue. In the next section we will extend this procedure to compute interior eigenvalues.

6.2.5 The Interior of the Spectrum

We are often interested not only in the smallest eigenvalues, but in eigenvalues in the interior of the spectrum. The $(n - j)$th eigenvalue for j small can be computed by the subspace preconditioned inverse iteration. But if j is large, say $j \gg \log_2 n$, this is prohibitive.

Instead, we will use the folded spectrum method [104] also mentioned in [79]. First we have to choose a shift σ. Then we compute the smallest eigenpair (λ_σ, v) of $M_\sigma = (M - \sigma I)^2$, through PINVIT. The eigenvector v is the eigenvector of M to the eigenvalue next to σ. The Rayleigh quotient $\mu(v, M) = v^T Mv/(v^T v)$ is the sought eigenvalue λ.

The condition number of the eigenvalue problem for $M_\sigma = (M - \sigma I)^2$ is 1, since M_σ is symmetric, see Lemma 2.1.12.

Lemma 6.2.8: Let $M \in \mathbb{R}^{n \times n}$ be symmetric positive definite and let v be an eigenvector of M, with $Mv = v\lambda$. Then v is also an eigenvector of M_σ. Further, the corresponding eigenvalue of M_σ is

$$\lambda_\sigma := (\lambda - \sigma)^2 .$$

If λ_σ is a simple eigenvalue of M_σ, then $\sigma + \sqrt{\lambda_\sigma}$ or $\sigma - \sqrt{\lambda_\sigma}$ is an eigenvalue of M.

Proof.

$$M_\sigma v = (M - \sigma I)^2 v = (M - \sigma I)(Mv - \sigma v) = (M - \sigma I)v(\lambda - \sigma) = v(\lambda - \sigma)^2$$

Since M is symmetric positive definite, there are n linearly independent eigenvectors v_i, for $i = 1, \ldots, n$. Each eigenvector v_i is an eigenvector of the matrix M_σ. Let v_j be the eigenvector corresponding to λ_σ, then (λ, v_j) is an eigenpair of M and λ and λ_σ fulfill the relation:

$$(\lambda - \sigma)^2 = \lambda_\sigma .$$

■

Now we use again $\mathcal{H}$-arithmetic, since the shifted and squared matrix

$$\tilde{M}_\sigma = (M -_{\mathcal{H}} \sigma I) *_{\mathcal{H}} (M -_{\mathcal{H}} \sigma I) \approx M_\sigma$$

is an $\mathcal{H}$-matrix like M. The $\mathcal{H}$-inversion of $\tilde{M}_\sigma$ is an approximate inverse of M_σ, so that we can use $(\tilde{M}_\sigma)^{-1}_{\mathcal{H}}$ as preconditioner. Also, the $\mathcal{H}$-Cholesky decomposition of $\tilde{M}_\sigma$ is an approximation to the Cholesky decomposition of M_σ and can also be used as preconditioner. The additional truncation in the computation of $\tilde{M}_\sigma$ does not disturb the computed eigenvalues, since the preconditioner does not need to be an exact inverse. The product $(M - \sigma I)^2 x_i$ in the computation of the residual needs a more accurate evaluation by the formula $(M - \sigma I)\left((M - \sigma I)x_i\right)$ instead of $\tilde{M}_\sigma x$.

Here $\mathcal{H}$-arithmetic is particularly advantageous, since $\mathcal{H}$-arithmetic enables us to compute an approximation to M_σ^{-1} or an approximate Cholesky decomposition with reasonable costs.

The smallest eigenvalue of M_σ is the square of the smallest by magnitude eigenvalue of $M - \sigma\mathcal{I}$. So the smallest eigenvalue of M_σ is often smaller than the smallest eigenvalue of $M - \sigma\mathcal{I}$. The squaring also increases the largest eigenvalue, meaning that the condition number of the linear system of equations with coefficients M_σ is increased by the shifting and squaring. A higher condition number leads to higher ranks in the admissible submatrices of the $\mathcal{H}$-inverse of M_σ, increasing the costs of the inversion and the application of the preconditioner. This is the main disadvantage of the folded spectrum method.

Further, if $\sigma + x$ and $\sigma - x$ are eigenvalues of M, then M_σ has a double eigenvalue and the folded spectrum method may fail to converge [79]. Therefore, we should avoid shifts

Algorithm 6.4: Inner Eigenvalues by Folded Spectrum Method and Hierarchical Subspace Preconditioned Inverse Iteration.

```
Input: $M \in \mathbb{R}^{n\times n}$, shift $\sigma$
Output: $X_p \in \mathbb{R}^{n\times d}$, $\mu \in \mathbb{R}^{d\times d}$, with $\|MX_p - X_p\mu\| \leq \epsilon$, $\Lambda(\mu)$ are
        approximations to the nearest eigenvalues to $\sigma$
1  Orthogonalize $X_0$;
2  $M_\sigma := (M -_{\mathcal{H}} \sigma\mathcal{I}) *_{\mathcal{H}} (M -_{\mathcal{H}} \sigma\mathcal{I})$;
3  $\mu_\sigma := X_0^T(M - \sigma\mathcal{I})(M - \sigma\mathcal{I})X_0$;
4  $R := (M - \sigma\mathcal{I})(M - \sigma\mathcal{I})X_0 - X_0\mu_\sigma$;
5  $T^{-1} = (M_\sigma)_{\mathcal{H}}^{-1}$;
6  $i := 1$;
7  while $\|R\|_F > \epsilon$ do
8      $i := i + 1$;
9      $X_i := X_{i-1} - T^{-1}R$;
10     Orthogonalize $X_i$;
11     $\mu_\sigma := X_i^T(M - \sigma\mathcal{I})(M - \sigma\mathcal{I})X_i$;
12     $R := (M - \sigma\mathcal{I})(M - \sigma\mathcal{I})X_i - X_i\mu_\sigma$;
13 end
14 $\mu := X_p^T M X_p$;
```

near the middle of two eigenvalues and ensure that we compute the whole subspace corresponding to the smallest eigenvalue.

The combination of the folded spectrum method and the preconditioned inverse iteration yields Algorithm 6.4.

Theorem 6.2.9: The for-loop of Algorithm 6.4 is the preconditioned inverse iteration applied to M_σ. If $M = M^T$ and $\sigma \notin \Lambda(M)$, then Theorem 6.2.4 holds for this algorithm.

Proof. We have M symmetric, meaning that $\lambda_i \in \mathbb{R}$. The eigenvalues λ_σ^i of $M_\sigma = (M - \sigma I)^2$ are $\lambda_\sigma^i = (\lambda_i - \sigma)^2$. Since $\sigma \neq \lambda_i$, we have $\lambda_\sigma^i > 0$ for all i. It follows that M_σ is symmetric positive definite.

The for-loop in Algorithm 6.4 is identical to the for-loop in Algorithm 6.3, except that M is replaced by M_σ. This means, that Algorithm 6.4 does a preconditioned inverse iteration. So Theorem 6.2.4 can be used here, too.

■

The numerical examples in the next section confirm that this algorithm works well.

Remark 6.2.10: The above described approach for computing the (inner) eigenvalues of an $\mathcal{H}$-matrix can also be applied to other data-sparse matrix formats. At the ENUMATH11 conference the author recently presented some first results on the computation of the eigenvalues of matrices in tensor train matrix format by using preconditioned inverse iteration together with the folded spectrum method [76].

6.2.6 Numerical Results

In this section we will show some numerical results confirming that Algorithm 6.3 computes the smallest eigenvalues in almost linear complexity. The numerical computations are done on Otto, see Section 2.7. We measure the required CPU time for the computation of the eigenvalues of matrices of different sizes. The convergence rate of the algorithm is determined by the gap between the largest eigenvalue in the computed invariant subspace and the smallest that is not in the subspace, see Theorem 6.2.4, Equation (6.6). We choose the size of the subspace and the shift for the last test so that the gap, see Equation (6.7), converges for n to infinity to a fixed number smaller than one. Another factor of big influence is the complexity of the inversion. We will measure and investigate the time for the inversion separately.

We choose the start vectors/matrices randomly. The computation of the preconditioner is the only operation with truncation and, since we use the adaptive inversion/adaptive Cholesky decomposition algorithm, we do not need to fix an accuracy for the $\mathcal{H}$-arithmetic. We stop the iteration if the residual is smaller than 10^{-4}. The eigenvalues then have (under special conditions) an accuracy of about 10^{-8}, see Lemma 6.2.7.

The influence of the sparsity and the idempotency constant, that are both slightly increasing in our example series, will be accounted for. We expect that the required CPU time grows asymptotically like

$$N(n_i) = C_{sp}C_{id}k^2 n_i \left(\log_2 n_i\right)^2 . \tag{6.11}$$

FEM matrices

First we test the algorithm with the FEM example series, from 8 inner discretization points to 512. These matrices have constant block-wise rank k. The results are shown in Table 6.1. The absolute resp. relative error is the maximum of the absolute resp. relative error for each eigenvalue. Figure 6.1 compares the CPU time for PINVIT$(q, 3)$, $q = 1$ or 3 with $N(n_i)$, see Equation (6.11). PINVIT$(1, 3)$ resp. PINVIT$(3, 3)$ means we compute the three smallest eigenvalues with preconditioned inverse iteration resp. linear optimal block preconditioned conjugate gradient method.

The results confirm our expectations. We should mention that the FEM matrices are sparse and sparse solvers like `eigs` in MATLAB are faster. In Table 6.1 and Figure 6.1 the time for the computation of the preconditioner is excluded. The inverse of a sparse matrix is, in general, not sparse. Here we know that the inverses of the FEM matrices

Preconditioner: $\mathcal{H}$-inversion						
Name	n	abs. error	rel. error	t in s	t_i/t_{i-1}	$\frac{N(n_i)}{N(n_{i-1})}$
FEM8	64	5.3725 e−10	7.0875 e−12	0.002		
FEM16	256	4.9434 e−10	6.3327 e−12	0.16	8.00	106.67
FEM32	1 024	4.6074 e−10	5.8530 e−12	0.11	6.75	27.08
FEM64	4 096	3.9327 e−10	4.9847 e−12	0.87	*8.07*	8.06
FEM128	16 384	4.4361 e−10	5.6194 e−12	5.39	*6.18*	5.44
FEM256	65 536	3.6599 e−10	4.6355 e−12	29.60	*5.49*	5.22
FEM512	262 144	4.0872 e−10	5.1765 e−12	203.00	*6.86*	6.51
Preconditioner: $\mathcal{H}$-Cholesky decomposition						
Name	n	abs. error	rel. error	t in s	t_i/t_{i-1}	$\frac{N(n_i)}{N(n_{i-1})}$
FEM8	64	5.5950 e−10	7.3811 e−12	0.001		
FEM16	256	4.5209 e−10	5.7914 e−12	0.16	16.00	106.67
FEM32	1 024	4.3180 e−10	5.4853 e−12	0.08	4.88	27.08
FEM64	4 096	4.1940 e−10	5.3159 e−12	0.43	5.51	8.06
FEM128	16 384	4.2118 e−10	5.3354 e−12	2.35	*5.46*	5.44
FEM256	65 536	4.1746 e−10	5.2874 e−12	12.50	*5.33*	5.22
FEM512	262 144	4.1886 e−10	5.3051 e−12	59.50	4.76	6.51
FEM1024	1 048 576	3.7377 e−10	4.7339 e−12	263.00	4.42	10.76

Table 6.1: Numerical results FEM-series, PINVIT(1, 3), $c = 0.2$, $\epsilon = 10^{-4}$; *italic entries* are larger than expected.

are still data-sparse in the $\mathcal{H}$-matrix sense.

Using the adaptive Cholesky decomposition as preconditioner reduces the costs of the algorithm, see Table 6.1. But even though the computation of the preconditioner is much cheaper, see Table 6.5, the whole algorithm is still about 20 times slower than the MATLAB function `eigs`. If we were to use the adaptive inverse as preconditioner, then the whole algorithm is about 2 000 times slower.

BEM matrices

The results for the BEM matrix series, see Table 6.2 and Figure 6.2, confirm again our expectation. Since the BEM matrices are dense, sparse solvers cannot be used here. The usage of dense solvers is no alternative, since the matrix BEM128 would need about 32 GB storage. We compare the computed eigenvalues with the ones computed by LAPACK [2]. Since BEM128 is too large, there is no comparison. The MATLAB function `eigs` is the fastest MATLAB built-in function for this problem, so we compare with eigs here, too. We are not able to read in the matrix BEM64 into MATLAB due to the

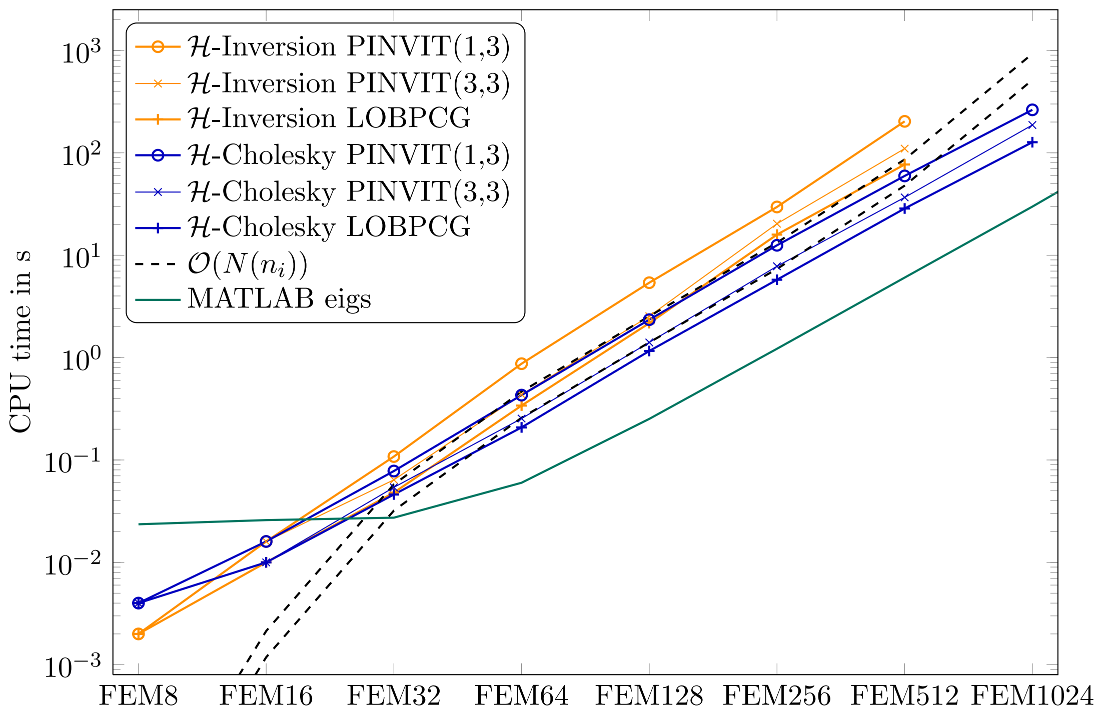


Figure 6.1: CPU time FEM-series, $c = 0.2$.

limitation of the size of m-files to 2 GB.

Inner eigenvalues

In this subsection we use the same matrices again. We shift them to demonstrate Algorithm 6.4. We choose the shift and the subspace dimension so that we have a large gap between the computed and the other eigenvalues. The gap is almost constant for both example series. The results are shown in Tables 6.3 and 6.4. There was not enough memory for the $\mathcal{H}$-inversion of the shifted FEM512 matrix and not enough fast memory for the $\mathcal{H}$-Cholesky decomposition of the shifted FEM512 matrix.

Adaptive $\mathcal{H}$-Inversion vs. Adaptive $\mathcal{H}$-Cholesky Decomposition

We finally investigate the computation of the preconditioner, which we excluded from the previous investigations. In Table 6.5 one can see that for the FEM-matrices, the storage and the computational complexity does not meat the expectation. Since the costs are still much lower than for dense arithmetic, we will not further investigate this small deviation that we found in the BEM-series, but only sporadically, see Table 6.6.

The computation of the preconditioner is responsible for at least half the cost of the whole

Preconditioner: $\mathcal{H}$-inversion						
Name	n	abs. error	rel. error	t in s	t_i/t_{i-1}	$\frac{N(n_i)}{N(n_{i-1})}$
BEM8	258	7.6641 e−08	1.3992 e−06	0.02		
BEM16	1 026	1.6227 e−07	6.1102 e−06	0.15	7.50	24.18
BEM32	4 098	8.9221 e−07	1.0276 e−04	6.69	*44.60*	39.56
BEM64	16 386	2.0029 e−06	2.9636 e−04	21.61	3.23	14.69
BEM128	65 538	—	—	140.66	6.51	14.66
Preconditioner: $\mathcal{H}$-Cholesky decomposition						
Name	n	abs. error	rel. error	t in s	t_i/t_{i-1}	$\frac{N(n_i)}{N(n_{i-1})}$
BEM8	258	5.8821 e−08	1.0740 e−06	0.01		
BEM16	1 026	1.4639 e−07	5.5112 e−06	0.11		24.18
BEM32	4 098	7.0811 e−07	5.1844 e−05	0.96	8.47	24.22
BEM64	16 386	1.8818 e−06	2.7845 e−04	7.36	7.67	23.99
BEM128	65 538	—	—	37.63	5.11	14.66

Table 6.2: Numerical results from BEM-series, PINVIT$(1,6)$, $c = 0.2$; *italic entries* are larger than expected.

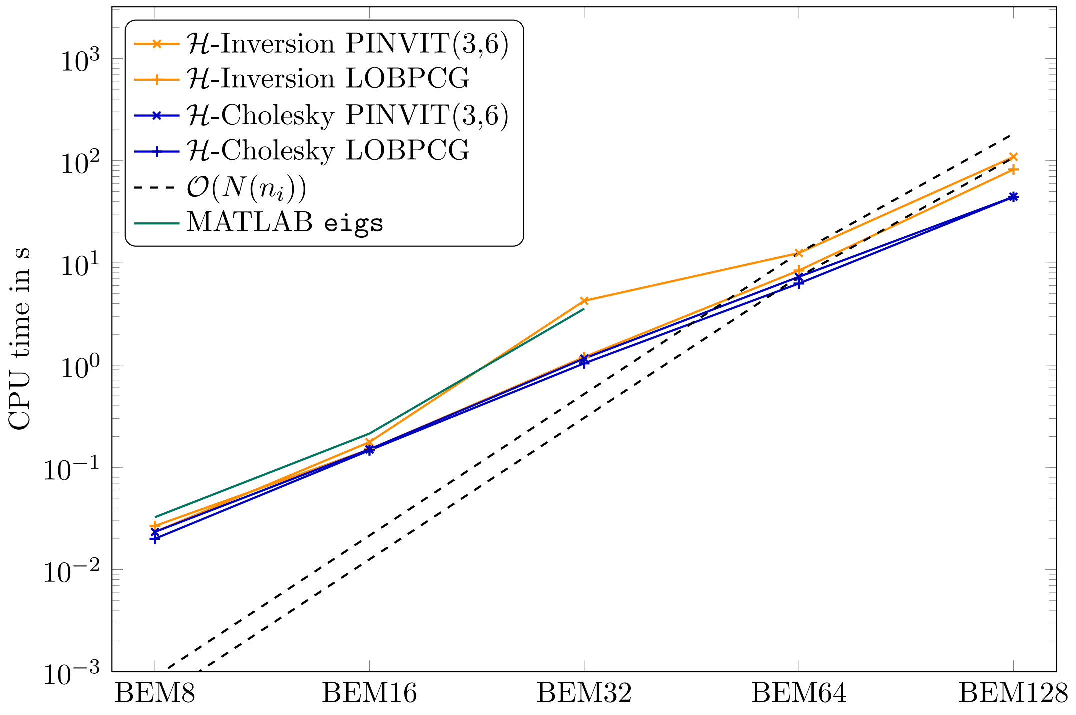


Figure 6.2: CPU time BEM-series, $c = 0.2$.

Preconditioner: $\mathcal{H}$-inversion

Name	n	abs. error	rel. error	t in s	t_i/t_{i-1}	$\frac{N(n_i)}{N(n_{i-1})}$
FEM8	64	2.2737 e−13	1.0582 e−15	<0.01		
FEM16	256	9.6634 e−12	5.5810 e−14	0.03		106.67
FEM32	1 024	3.2458 e−11	1.6613 e−13	0.08	2.67	27.08
FEM64	4 096	1.0232 e−12	5.7695 e−15	0.67	*8.38*	8.06
FEM128	16 384	1.1085 e−12	5.6192 e−15	4.61	*6.88*	5.44
FEM256	65 536	5.4570 e−12	3.0080 e−14	36.64	*7.94*	5.22

Preconditioner: $\mathcal{H}$-Cholesky decomposition

Name	n	abs. error	rel. error	t in s	t_i/t_{i-1}	$\frac{N(n_i)}{N(n_{i-1})}$
FEM8	64	1.7053 e−13	7.9365 e−16	<0.01		
FEM16	256	7.4465 e−12	4.3006 e−14	0.02		106.67
FEM32	1 024	3.2571 e−11	1.6671 e−13	0.04	2.00	27.09
FEM64	4 096	8.5265 e−13	4.8080 e−15	0.21	5.25	8.06
FEM128	16 384	9.6634 e−13	5.4419 e−15	1.52	*7.24*	5.44
FEM256	65 536	6.9065 e−12	3.4995 e−14	10.48	*6.90*	5.22
FEM512	262 144	9.3138 e−11	4.8173 e−13	54.81	5.23	6.51

Table 6.3: Numerical results from shifted FEM-series, PINVIT$(1, 3)$, $c = 0.2$, $\epsilon = 10^{-4}$; *italic entries* are larger than expected.

Preconditioner: $\mathcal{H}$-inversion

Name	n	abs. error	rel. error	t in s	t_i/t_{i-1}	$\frac{N(n_i)}{N(n_{i-1})}$
BEM8	258	3.2609 e−06	1.1631 e−06	0.05		
BEM16	1 026	3.3596 e−07	9.2508 e−08	0.27	5.40	24.18
BEM32	4 098	3.0287 e−04	9.1472 e−05	0.72	2.67	24.22
BEM64	16 386	3.4399 e−05	1.0458 e−05	17.46	*24.25*	23.99
BEM128	65 538	—	—	25.59	1.47	14.66

Preconditioner: $\mathcal{H}$-Cholesky decomposition

Name	n	abs. error	rel. error	t in s	t_i/t_{i-1}	$\frac{N(n_i)}{N(n_{i-1})}$
BEM8	258	1.9307 e−06	6.8865 e−07	0.08		
BEM16	1 026	5.5875 e−07	1.5386 e−07	0.26	3.25	24.18
BEM32	4 098	3.4877 e−04	1.0533 e−04	0.72	2.77	24.22
BEM64	16 386	2.2995 e−05	6.9910 e−06	8.40	11.67	23.99
BEM128	65 538	—	—	48.77	5.81	14.66

Table 6.4: Numerical results from shifted BEM-series, PINVIT$(1, 6)$, $c = 0.2$, $\epsilon = 10^{-4}$; *italic entries* are larger than expected.

$\mathcal{H}$-inversion						
Name	n	$s(M^{-1})$ in kB	s_i/s_{i-1}	t in s	t_i/t_{i-1}	$\frac{N(n_i)}{N(n_{i-1})}$
FEM8	64	25		0.01		
FEM16	256	246	10.02	0.04	4.00	106.67
FEM32	1 024	2 089	8.48	0.38	9.50	27.08
FEM64	4 096	16 558	7.93	7.21	*18.97*	8.,06
FEM128	16 384	116 629	7.04	57.77	*8.01*	5.44
FEM256	65 536	755 768	6.48	420.42	*7.28*	5.22
FEM512	262 144	4 542 690	6.01	2 891.06	*6.88*	6.51

$\mathcal{H}$-Cholesky decomposition						
Name	n	$s(L)$ in kB	s_i/s_{i-1}	t in s	t_i/t_{i-1}	$\frac{N(n_i)}{N(n_{i-1})}$
FEM8	64	25		<0.01		
FEM16	256	161	6.58	<0.01		106.67
FEM32	1 024	962	5.96	0.04		27.08
FEM64	4 096	5 401	5.61	0.25	6.25	8.06
FEM128	16 384	29 077	5.38	1.70	*6.80*	5.44
FEM256	65 536	152 120	5.23	10.03	*5.90*	5.22
FEM512	262 144	785 751	5.17	56.34	5.62	6.51
FEM1024	1 048 576	3 974 881	5.06	332.05	5.89	10.76

Table 6.5: Required storage and CPU-time for adaptive $\mathcal{H}$-inversion and adaptive $\mathcal{H}$-Cholesky decomposition of different FEM-matrices, $c = 0.2$; *italic entries* are larger than expected.

$\mathcal{H}$-inversion						
Name	n	$s(M^{-1})$ in kB	s_i/s_{i-1}	t in s	t_i/t_{i-1}	$\frac{N(n_i)}{N(n_{i-1})}$
BEM8	258	542		0.31		
BEM16	1 026	5 443	10.04	3.12	10.06	24.18
BEM32	4 098	47 825	8.79	28.62	9.17	24.22
BEM64	16 386	470 293	9.83	637.62	22.28	23.99
BEM128	65 538	4 597 070	9.77	8 625.12	13.53	14.66
$\mathcal{H}$-Cholesky decomposition						
Name	n	$s(L)$ in kB	s_i/s_{i-1}	t in s	t_i/t_{i-1}	$\frac{N(n_i)}{N(n_{i-1})}$
BEM8	258	453		0.11		
BEM16	1 026	3 114	6.87	0.39	3.55	24.18
BEM32	4 098	21 117	6.78	3.42	8.77	24.22
BEM64	16 386	127 142	6.02	25.97	7.59	23.99
BEM128	65 538	735 563	5.79	189.84	7.31	14.66

Table 6.6: Required storage and CPU-time for adaptive $\mathcal{H}$-inversion and adaptive $\mathcal{H}$-Cholesky decomposition of different BEM-matrices, $c = 0.2$; *italic entries* are larger than expected.

algorithm. The only parameter directly effecting these costs is the c in Equation (6.5), that we choose relatively large with $c = 0.2$.

The adaptive $\mathcal{H}$-Cholesky decomposition is much cheaper than the adaptive $\mathcal{H}$-inversion as in some cases the Cholesky decomposition needs only 2% of the time of the inversion. The Cholesky factors require less storage than the $\mathcal{H}$-inverse. This is beneficial in two ways. On the one hand, less storage means that we can handle larger problems, e. g. shifted FEM512. On the other hand, the effort required for the application of the preconditioner is reduced. This seems to be the main explanation for the lower CPU times we see in Tables 6.1, 6.2, and 6.3, since the numbers of iterations are similar.

To summarize, one should use the adaptive $\mathcal{H}$-Cholesky decomposition instead of the adaptive $\mathcal{H}$-inversion.

Higher dimensional problems

Elliptic eigenvalue problems for dense matrices occur, e.g., in the Hartree-Fock equations in electronic structure calculations, see [23]. A first step towards this problems from computational chemistry is the investigation of 3D problems.

The sparsity constant of $\mathcal{H}$-matrices originating from 3D problems is much larger and, as such, more computational effort is necessary. In Table 6.7 we use the FEM discretization of the 3D Laplacian to test our algorithm for 3D problems. Since the PINVIT$(1, 4)$ does

Preconditioner: $\mathcal{H}$-Cholesky decomposition

Name	n	abs. error	rel. error	t in s	$\frac{t_i}{t_{i-1}}$	$\frac{N(n_i)}{N(n_{i-1})}$	t_{eigs}
FEM3D4	64	1.194 e−12	4.450 e−15	<0.01			0.02
FEM3D8	512	1.933 e−12	3.738 e−15	0.03		3 060.00	0.03
FEM3D16	4 096	6.651 e−12	1.305 e−14	0.45	15.00	164.95	0.18
FEM3D32	32 768	1.433 e−11	7.347 e−15	7.40	16.44	235.22	4.93
FEM3D64	262 144	1.039 e−10	5.400 e−14	194.89	26.34	484.91	140.56
FEM3D128	2 079 152	9.756 e−12	1.648 e−13	1 929.81	9.90	14.29	—

Table 6.7: Numerical results from 3D FEM-series, PINVIT$(3, 4)$, $c = 0.2$, $\epsilon = 10^{-4}$.

not converge, we use PINVIT$(3, 4)$/LOBPCG instead. The estimate $N(n_i)/N(n_{i-1})$ contains strong growth in the constants C_{sp} and C_{id}. The last column shows the time the MATLAB function `eigs` requires. We see that for large matrices `eigs` is more expensive.

If we go on to higher dimensional problems with $d \gg 3$, then the so called "curse of dimensionality" forces the usage of data-sparse tensor structures, as is done in [59] and [67]. The structure of $\mathcal{H}$-matrices is not sufficient for these problems, so problem tailored special eigensolvers are more efficient. Use of tensor trains, see [85], is one way to overcome this curse. As mentioned in Remark 6.2.10 this will be investigated in [76].

6.2.7 Conclusions

We have seen that the preconditioned inverse iteration can be used to compute the smallest eigenvalues of hierarchical matrices. Further, one can use the ideas of the folded spectrum method together with the efficient $\mathcal{H}$-arithmetic to compute inner eigenvalues of an $\mathcal{H}$-matrix using PINVIT.

This can be used to compute inner eigenvalues of sparse matrices as well as of data-sparse matrices like boundary element matrices.

For sparse matrices the handling as $\mathcal{H}$-matrix and the computation of the eigenvalues by preconditioned inverse iteration for $\mathcal{H}$-matrices is not competitive, since the $\mathcal{H}$-inversion is particularly too expensive. If we have the $\mathcal{H}$-inverse or $\mathcal{H}$-Cholesky decomposition already available, when we have, for instance, to solve some systems $Mx = b$ as well, then the approach presented here is competitive to sparse eigensolvers.

In the case of data-sparse matrices that are not sparse, the limitation of storage limits the use of dense eigensolvers. The data-sparse $\mathcal{H}$-arithmetic together with preconditioned inverse iteration enables us to solve larger eigenvalue problems that do not fit in the storage otherwise.

In the next chapter we will compare the different algorithms presented in the previous chapters.

CHAPTER

SEVEN

COMPARISON OF THE ALGORITHMS AND NUMERICAL RESULTS

Contents

The aim of this last but one chapter is the comparison of the algorithms. First, we compare them by their theoretical properties. Afterwards, numerical computations will be presented comparing the algorithms by application.

7.1 Theoretical Comparison

The algorithms differ in the computed eigenvalues (the smallest, some inner, or all eigenvalues), in complexity, and in their input matrices ($\mathcal{H}$-matrices or only $\mathcal{H}_\ell$-matrices).

QR Algorithm for Dense Matrices The QR algorithm for dense matrices is a well known and highly optimized algorithm for the computation of eigenvalues. This algorithm is used as a reference example. The algorithm computes all eigenvalues in an unknown order and is also able to compute the eigenvectors. The main drawback is that we have to transform the $\mathcal{H}$-matrix into a dense matrix, which costs $\mathcal{O}(kn^2)$ flops and requires n^2 storage entries. The algorithm itself is of cubic complexity. In [46] the number of flops is given with $25\,n^3$ (including eigenvectors) resp. $10\,n^3$ (only eigenvalues).

$\mathcal{H}$-LR Cholesky Algorithm The algorithm from Chapter 4 is efficient only for $\mathcal{H}_\ell(k)$-matrices ($\mathcal{O}(k^2n^2\,(\log_2 n)^4)$ flops and $\mathcal{O}(k^2n\,(\log_2 n)^2)$ storage), where the algorithm stays within the set of $\mathcal{H}_\ell(k\ell)$-matrices. Besides the current iterate, there is no other

		symmetric $\mathcal{H}$-matrix		
eigenvalue(s)		smallest	inner	all
	memory	$\mathcal{O}(C_{sp}kn\log_2 n)$		
dense LAPACK		✓	✓	✓
`dsyev`	flops	$\mathcal{O}(n^3)$	$\mathcal{O}(n^3)$	$\mathcal{O}(n^3)$
	memory	n^2	n^2	n^2
$\mathcal{H}$LR Cholesky		✓	✓	✓
algorithm	flops	$\mathcal{O}(n^4)$	$\mathcal{O}(n^4)$	$\mathcal{O}(n^4)$
	memory	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
Slicing		o	o	o
algorithm	flops	—	—	—
	memory	—	—	—
$\mathcal{H}$-PINVIT		✓	✓	o
	flops	$\mathcal{O}(k^2n\,(\log_2 n)^2)$	$\mathcal{O}(k^2n\,(\log_2 n)^2)$	—
	memory	$\mathcal{O}(kn\log_2 n)$	$\mathcal{O}(kn\log_2 n)$	—
		symmetric $\mathcal{H}_\ell$-matrix		
eigenvalue(s)		smallest	inner	all
	memory	$2kn\log_2 n + nn_{\min}$		
dense LAPACK		✓	✓	✓
`dsyev`	flops	$\mathcal{O}(n^3)$	$\mathcal{O}(n^3)$	$\mathcal{O}(n^3)$
	memory	n^2	n^2	n^2
$\mathcal{H}$LR Cholesky		✓	✓	✓
algorithm	flops	$\mathcal{O}(k^2n^2\,(\log_2 n)^4)$	$\mathcal{O}(k^2n^2\,(\log_2 n)^4)$	$\mathcal{O}(k^2n^2\,(\log_2 n)^4)$
	memory	$\mathcal{O}(kn\,(\log_2 n)^2)$	$\mathcal{O}(kn\,(\log_2 n)^2)$	$\mathcal{O}(kn\,(\log_2 n)^2)$
Slicing		✓	✓	✓
algorithm	flops	$\mathcal{O}(k^2n\,(\log_2 n)^4)$	$\mathcal{O}(k^2n\,(\log_2 n)^4)$	$\mathcal{O}(k^2n^2\,(\log_2 n)^4)$
	memory	$\mathcal{O}(kn\,(\log_2 n)^2)$	$\mathcal{O}(kn\,(\log_2 n)^2)$	$\mathcal{O}(kn\,(\log_2 n)^2)$
$\mathcal{H}$-PINVIT		✓	✓	o
	flops	$\mathcal{O}(k^2n\,(\log_2 n)^2)$	$\mathcal{O}(k^2n\,(\log_2 n)^2)$	—
	memory	$\mathcal{O}(kn\log_2 n)$	$\mathcal{O}(kn\log_2 n)$	—

Table 7.1: Theoretical properties of the algorithms.

matrix that has to be stored. Due to the shift strategy, we expect to find the smallest eigenvalues first. The algorithm does not compute eigenvectors. For $\mathcal{H}$-matrices the algorithm is not structure preserving, meaning that $\mathcal{O}(n^2)$ storage entries and up to $\mathcal{O}(n^4)$ flops are necessary.

Slicing the Spectrum Algorithm In Chapter 5 we have seen how to use the $\mathcal{H}$-LDL^T factorization for the computation of eigenvalues by bisectioning. The algorithm computes some or all eigenvalues of an $\mathcal{H}_\ell$-matrix in $\mathcal{O}(k^2 n\,(\log_2 n)^4)$ flops per eigenvalue. Therefore, the algorithm requires an additional storage of only $\mathcal{O}(k^2 n\,(\log_2 n)^2)$ to the $\mathcal{H}_\ell$-matrix M for storing the shifted matrix resp. the $\mathcal{H}$-LDL^T factorization. The algorithm has a simple parallelization, since the computations for different eigenvalues are independent after the first steps. The algorithm does not compute eigenvectors.

If the algorithm is applied to $\mathcal{H}$-matrices, then the truncated $\mathcal{H}$-LDL^T factorization sometimes leads to a wrong inertia. This means that some of the computed approximations are far away from the eigenvalues.

$\mathcal{H}$-PINVIT The preconditioned inverse iteration, see Chapter 6, computes the smallest eigenvalue/s and the corresponding eigenvector resp. an invariant subspace containing the corresponding eigenvectors. The complexity of the method is determined by the $\mathcal{H}$-inversion procedure, which is in $\mathcal{O}(k^2 n^2\,(\log_2 n)^2)$. If PINVIT is combined with the folded spectrum method, then inner eigenvalues can also be computed. The storage requirements are as low as for the slicing algorithm, since one has to store M and the approximate $\mathcal{H}$-inverse resp. $\mathcal{H}$-Cholesky decomposition.

Using a subspace of dimension n in order to compute all eigenvalues is too expensive. This means that the algorithm should only be used for the computation of some eigenvalues.

The facts listed in the last paragraphs are summarized in Table 7.1. In the next section the algorithms are compared by numerical experiments.

7.2 Numerical Comparison

In the last section we have compared the algorithms by their theoretical properties. The constant hidden in the complexity estimation is, however, also of importance. In this section we will compare the algorithms by measuring the required CPU time and the achieved accuracy in the computed eigenvalues. As a reference, we use the exact eigenvalues for the FEM example series and the eigenvalues computed by the LAPACK function `dsyev` for the other matrices. For PINVIT we use the variant LOBPCG and sum the time for the computation of the preconditioner and the iteration process. We use in this section the Cholesky decomposition as preconditioner.

			$\mathcal{H}$-PINVIT		LDL^T slicing algorithm	
Name	n	t_{eigs} in s	t in s	rel. error	t in s	rel. error
bcsstk08	1 074	0.09	1.25	8.6150 e−05	50.79	6.2526 e−06
bcsstk38	8 032	0.29	10.42	8.3160 e−04	270.35	1.7327 e+00[1]
msc10848	10 848	0.80	250.78	1.3138 e−04	4 054.75	1.7732 e−01[2]
msc23052	23 052	2.00	70.95	9.8048 e−07	21 109.30	7.1287 e−01[3]
BEM32	4 098	7.35	3.88	3.1573 e−05	215.46	5.4523 e−03
BEM64	16 386	too large	33.07	4.0337 e−05	1770.97	4.7368 e−02

Table 7.2: Comparison between Slicing the spectrum, $\mathcal{H}$-PINVIT, and MATLAB function `eigs` for the computation of three smallest eigenvalues of different matrices.

Smallest Eigenvalues First we have a look at the smallest eigenvalues. Since the LR Cholesky algorithm computes the eigenvalues in an unknown ordering, we have to compute all eigenvalues. This takes $\mathcal{O}(n^2)$ flops and is more expensive than the other algorithms of almost linear complexity. Figure 7.1 shows that the preconditioned inverse iteration is superior to the LDL^T slicing algorithm if we compute the eigenvalues for the FEM example matrices. For the $\mathcal{H}_\ell$-matrices, both algorithms require almost the same time, see Figure 7.2. If the approximate inverse is already computed, then $\mathcal{H}$-PINVIT is the method of choice.

The MATLAB built-in function `eigs` is very good for sparse matrices, but too expensive for the dense $\mathcal{H}_\ell$-matrices, since n^2 storage and $\mathcal{O}(n^2)$ flops are required. Further results for different matrices are shown in Table 7.2. We observe that the computation of small eigenvalues to a high relative accuracy takes a lot of bisectioning steps and makes the LDL^T slicing algorithm expensive. Otherwise we get only rough approximations with large relative errors and small absolute errors.

Inner Eigenvalues The inversion of the badly conditioned matrix M_σ is more expensive than the inversion of M. This makes the slicing algorithm more competitive. Further, the Cholesky decomposition of M_σ failed for H10–H15, since M_σ has a large condition number. The Cholesky decomposition does not preserve positive definiteness as suggested in [7]. As such, some of the $M_{22} -_{\mathcal{H}} L_{21}L_{21}^T$ blocks in the recursion are not positive definite due to the truncation. The LDL^T slicing algorithm is a little cheaper by comparable accuracy than the $\mathcal{H}$-PINVIT, see Figure 7.3.

[1]The absolute error is 1.1650 e−16.
[2]The absolute error is 2.5703 e−11.
[3]The absolute error is 3.3547 e−12.

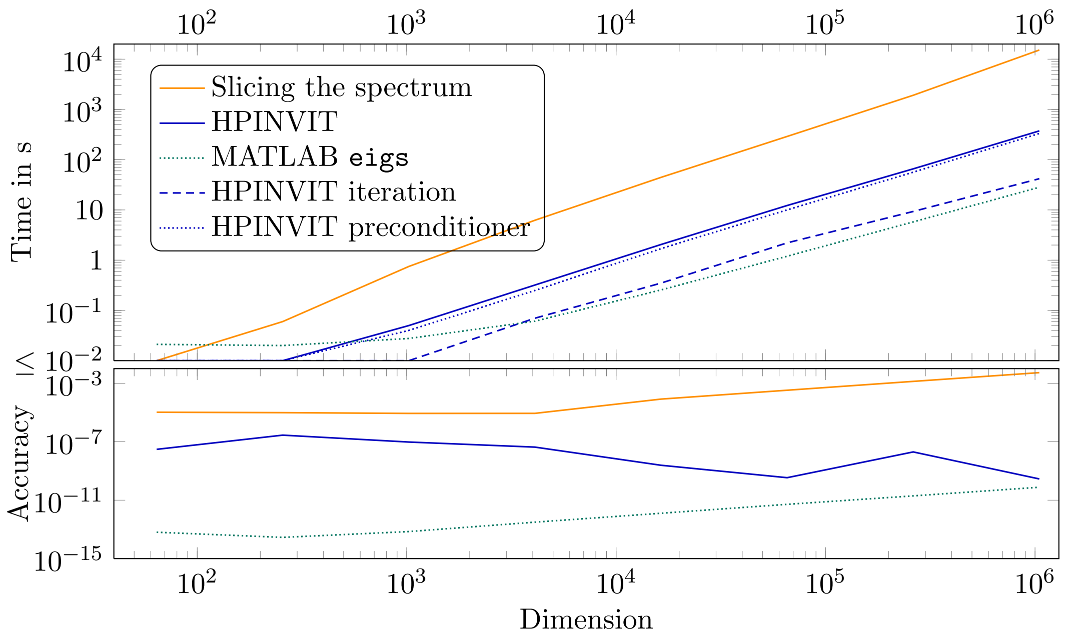


Figure 7.1: Comparison between Slicing the spectrum, $\mathcal{H}$-PINVIT, and MATLAB function `eigs` for the computation of the three smallest eigenvalues of FEM matrices.

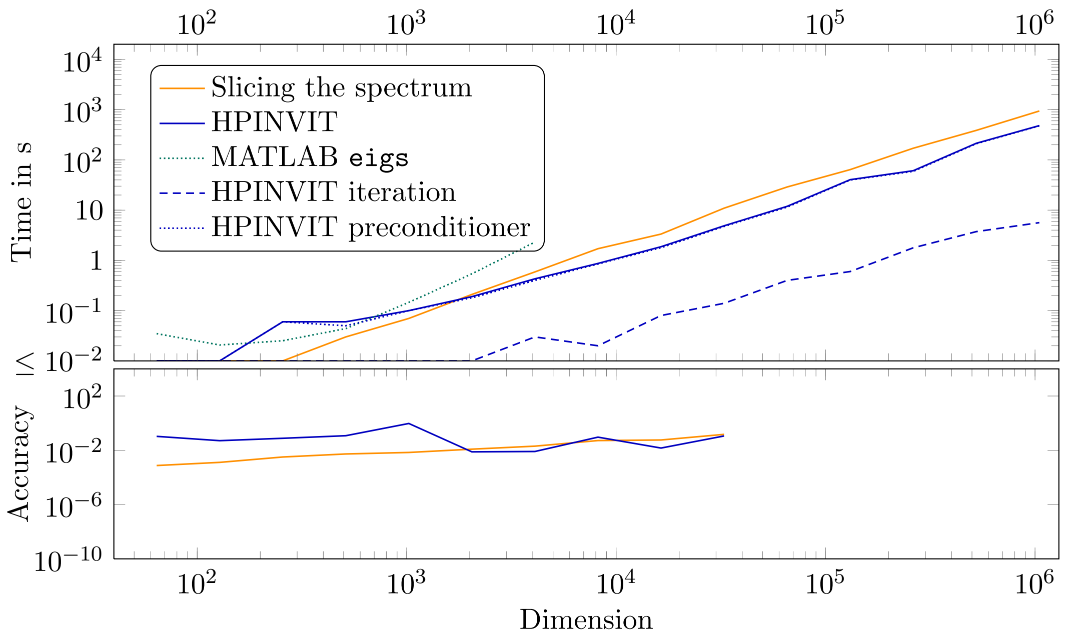


Figure 7.2: Comparison between Slicing the spectrum, $\mathcal{H}$-PINVIT, and MATLAB function `eigs` for the computation of the three smallest eigenvalues of $\mathcal{H}_\ell(1)$-matrices.

		LDL^T slicing algorithm		projection method			
Name	n	t in s	rel. error	t in s	rel. error	a	b
FEM8	64	0.03	4.6472 e−09	0.02	8.1609 e−16	535	592
FEM64	4 096	6.60	1.0337 e−09	46.97	2.6031 e−11	2120	2140
FEM128	16 384	45.89	3.7249 e−10	300.18	3.8128 e−10	2120	2140
H10 r1	32 768	7.42	1.8088 e−06	23.57	4.8686 e−15	0.675	0.8
H12 r1	131 072	77.08	—	251.67	—	0.53	0.57
BEM8	258	0.69	4.1661 e−03	1.52	7.1203 e−07	0.55	0.57

Table 7.3: Comparison between slicing the spectrum and projection method for the computation of some inner eigenvalues of different matrices.

			LR Cholesky algorithm		LDL^T slicing algorithm	
Name	n	t_{dsyev} in s	t in s	rel. error	t in s	rel. error
FEM32	1 024	0.7	393.99	5.2716 e−06	217.31	5.7118 e−08
FEM128	16 384	2 819.50	out of memory		35 613.28	4.3089 e−03
bcsstk08	1 074	0.95	1 384.64	1.1053 e+01[1]	41.72	1.9853 e+01[2]
					8 086.55	5.7065 e−04

Table 7.4: Comparison between slicing the spectrum, LR Cholesky algorithm, and LAPACK function `dsyev` for the computation of all eigenvalues of different matrices.

Projection Method vs. LDL^T Slicing Algorithm In Table 7.3 we compare the projection method [60], see Subsection 2.6.1, and the LDL^T slicing algorithm. We compute all eigenvalues in the interval (a, b). The LDL^T slicing algorithm is 3 to 7 times faster, but the projection method give more accurate results. For H12 r1 we do not have the exact eigenvalues or results given by the LAPACK function `dsyev`.

All Eigenvalues In this paragraph we only compare the LR Cholesky algorithm and the LDL^T slicing algorithm. For the $\mathcal{H}_\ell(1)$-matrices, both algorithms have almost the same run time, but the LDL^T slicing algorithm is more accurate, see Figure 7.4. We also see in the figure that the green dotted line and the orange line will intersect. For large $\mathcal{H}_\ell(1)$-matrices the LDL^T slicing algorithm is faster than the LAPACK function `dsyev`.

That is not the case for more general structured matrices, see Table 7.4. The matrix bcsstk08 has small eigenvalues, which are only approximated with large relative errors. Both algorithms become expensive if small eigenvalues are required with high relative

[1]The absolute error is 4.0803 e−06.
[2]The absolute error is 3.9288 e−04.

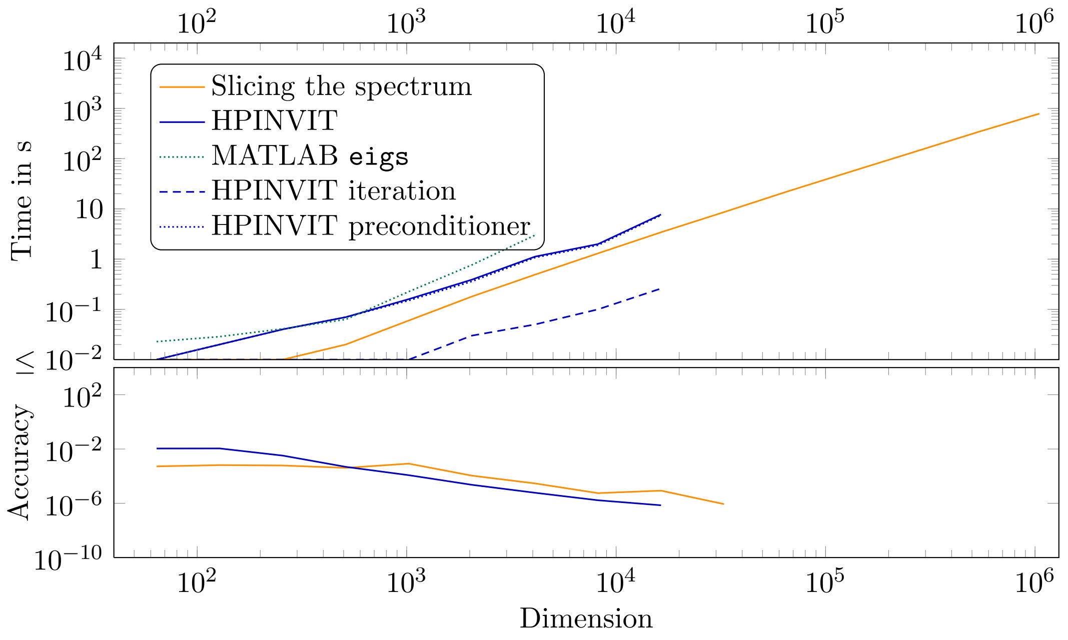


Figure 7.3: Comparison between Slicing the spectrum, $\mathcal{H}$-PINVIT, and MATLAB function `eigs` for the computation of of $\lambda_{n/4+5}, \ldots, \lambda_{n/4+14}$ of $\mathcal{H}_\ell(1)$-matrices.

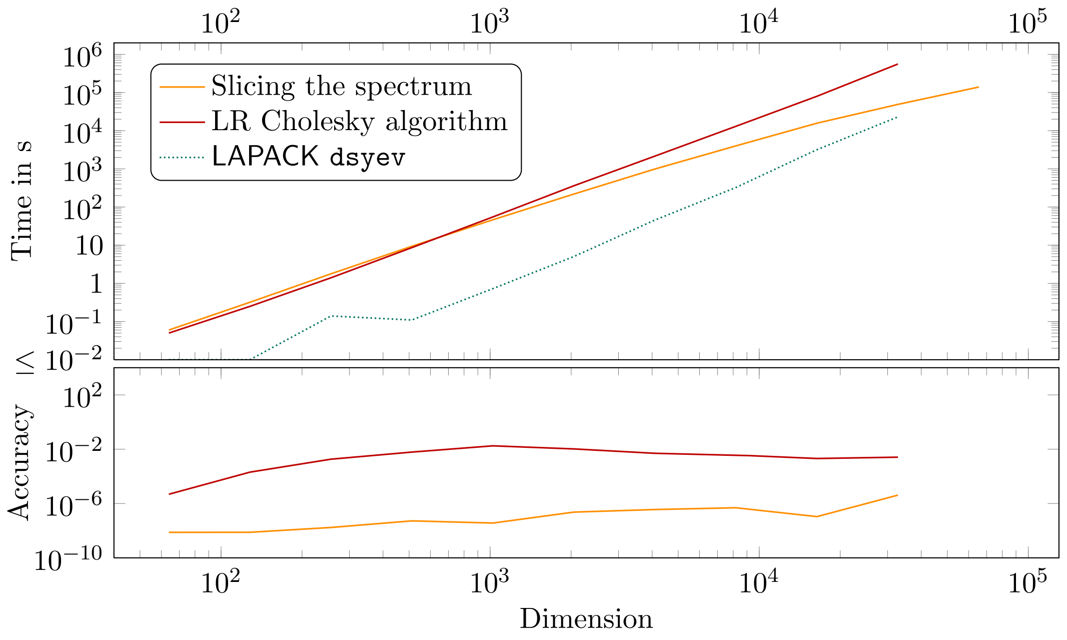


Figure 7.4: Comparison between Slicing the spectrum, LR Cholesky algorithm, and LAPACK function `dsyev` for the computation of all eigenvalues of $\mathcal{H}_\ell(1)$-matrices.

accuracy. We run the LDL^T slicing algorithm twice for bcsstk08 with different target accuracies. The second run achieves a smaller relative error, but takes extremely long time to run.

In this chapter we have compared the algorithms presented in the previous chapters. We have seen that, depending on the task, smallest, inner, or all eigenvalues and $\mathcal{H}_\ell$- or general $\mathcal{H}$-matrix, the algorithms perform different. For dense matrices the LAPACK solvers are the fastest for computing all eigenvalues. The LDL^T slicing algorithm is competitive only for $\mathcal{H}_\ell$-matrices. If the dense matrix does not fit into the RAM, then the usage of $\mathcal{H}$-matrices, which require fewer storage, should be considered. For the computation of some eigenvalues the MATLAB function `eigs` is superior only for sparse matrices. The preconditioned inverse iteration and LDL^T slicing algorithm are the best algorithms for the computation of some eigenvalues of dense, but data-sparse, matrices.

CHAPTER

EIGHT

CONCLUSIONS

We have seen different algorithms for computing the eigenvalues of symmetric hierarchical matrices. The structure of $\mathcal{H}$-matrices is data-sparse, which enables us to store them in a linear-polylogarithmic amount of storage. We further exploit the structure to reduce the complexity of the eigenvalue algorithms.

We investigated the subset of $\mathcal{H}_\ell$-matrices. The weak admissible condition used for the $\mathcal{H}_\ell$-matrices produce a simple hierarchical structure. This simple structure yields cheaper arithmetic operations. Besides this, there are exact inversions and LDL^T factorizations for $\mathcal{H}_\ell$-matrices with almost the same block-wise ranks.

In Chapter 3 we investigated a new algorithm for the computation of the QR decomposition for hierarchical matrices. This algorithm was compared with two existing algorithms for the $\mathcal{H}$-QR decomposition. In the next chapter we try to use this QR decomposition to build a QR algorithm for $\mathcal{H}$-matrices resp. a LR Cholesky algorithm. We realized that this leads to growing block-wise ranks destroying the data-sparse structure. With the new proof for Theorem 4.3.1, we get the tool to explain this behavior. We further used this to show that the structure of $\mathcal{H}_\ell$-matrices is almost preserved under LR Cholesky transformation.

In Chapter 5 we presented a different approach to the eigenvalue problem using a bisectioning method based on the LDL^T factorization. The structure of $\mathcal{H}_\ell$-matrices is simple enough to compute the LDL^T factorization for any shift without truncation in $\mathcal{O}(k^2 n\,(\log_2 n)^4)$ flops. The presented algorithm for the eigenvalue problem computes one eigenvalue in

$$\mathcal{O}(k^2 n\,(\log_2 n)^4 \log(\|M\|_2/\epsilon_{\mathrm{ev}})) \quad \text{flops.}$$

The computation of all n eigenvalues is n times more expensive. We showed that this LDL^T slicing algorithm is very well parallelizable. Generalization to $\mathcal{H}$-matrices is difficult. The LDL^T factorization for $\mathcal{H}$-matrices requires truncation to be in linear-polylogarithmic complexity for positive definite matrices. For arbitrary shifts there is no known bound on the block-wise ranks in fixed accuracy arithmetic. We also run the LDL^T slicing algorithm for $\mathcal{H}$-matrices. The algorithm gives us approximations of the eigenvalues, but we did not have bounds on the errors or on the run time.

Afterwards, we had a look at vector iterations. The preconditioned inverse iteration has nice convergence properties. PINVIT should be used for the computation of the smallest eigenvalues. This, in combination with folded spectrum method, also yields a method for computing some inner eigenvalues. However, the LDL^T slicing algorithm is competitive for the computation of inner eigenvalues.

Finally, we compared the different algorithms. For small problems, the LAPACK eigensolvers are the most efficient. For larger matrices, the storage limits the usage of dense matrices, which require n^2 storage. The $\mathcal{H}$-matrices require only a linear-polylogarithmic amount of storage, so that larger problems can be handled. We also compared the new LDL^T slicing algorithm with the already existing projection method. We saw that the projection method is more expensive.

Exploiting the structure accelerates the computation of the eigenvalues of $\mathcal{H}$- and $\mathcal{H}_\ell$-matrices.

THESES

1. This thesis is on the numerical computation of eigenvalues of symmetric hierarchical matrices. The numerical algorithms used for this computation are derivations of the LR Cholesky algorithm, the preconditioned inverse iteration, and a bisection method based on LDL^T factorizations.

2. The investigation of QR decompositions for $\mathcal{H}$-matrices leads to a new QR decomposition. It has some properties that are superior to the existing ones, which is shown by experiments using the $\mathcal{H}$QR decompositions to build a QR (eigenvalue) algorithm for $\mathcal{H}$-matrices does not progress to a more efficient algorithm than the LR Cholesky algorithm.

3. The implementation of the LR Cholesky algorithm for hierarchical matrices together with deflation and shift strategies yields an algorithm that require $\mathcal{O}(n)$ iterations to find all eigenvalues. Unfortunately, the local ranks of the iterates show a strong growth in the first steps. These $\mathcal{H}$-fill-ins makes the computation expensive, so that $\mathcal{O}(n^3)$ flops and $\mathcal{O}(n^2)$ storage are required.

4. Theorem 4.3.1 explains this behavior and shows that the LR Cholesky algorithm is efficient for the simple structured $\mathcal{H}_\ell$-matrices.

5. There is an exact LDL^T factorization for $\mathcal{H}_\ell$-matrices and an approximate LDL^T factorization for $\mathcal{H}$-matrices in linear-polylogarithmic complexity. This factorizations can be used to compute the inertia of an $\mathcal{H}$-matrix. With the knowledge of the inertia for arbitrary shifts, one can compute an eigenvalue by bisectioning. The slicing the spectrum algorithm can compute all eigenvalues of an $\mathcal{H}_\ell$-matrix in linear-polylogarithmic complexity. A single eigenvalue can be computed in $\mathcal{O}(k^2 n \,(\log_2 n)^4)$.

6. Since the LDL^T factorization for general $\mathcal{H}$-matrices is only approximative, the accuracy of the LDL^T slicing algorithm is limited. The local ranks of the LDL^T factorization for indefinite matrices are generally unknown, so that there is no statement on the complexity of the algorithm besides the numerical results in Table 5.7.

7. The preconditioned inverse iteration computes the smallest eigenvalue and the corresponding eigenvector. This method is efficient, since the number of iterations is independent of the matrix dimension.

8. If other eigenvalues than the smallest are searched, then preconditioned inverse iteration can not be simply applied to the shifted matrix, since positive definiteness is necessary. The squared and shifted matrix $(M - \mu I)^2$ is positive definite. Inner eigenvalues can be computed by the combination of folded spectrum method and PINVIT. Numerical experiments show that the approximate inversion of $(M-\mu I)^2$ is more expensive than the approximate inversion of M, so that the computation of the inner eigenvalues is more expensive.

9. We compare the different eigenvalue algorithms. The preconditioned inverse iteration for hierarchical matrices is better than the LDL^T slicing algorithm for the computation of the smallest eigenvalues, especially if the inverse is already available. The computation of inner eigenvalues with the folded spectrum method and preconditioned inverse iteration is more expensive. The LDL^T slicing algorithm is competitive to $\mathcal{H}$-PINVIT for the computation of inner eigenvalues.

10. In the case of large, sparse matrices, specially tailored algorithms for sparse matrices, like the MATLAB function `eigs`, are more efficient.

11. If one wants to compute all eigenvalues, then the LDL^T slicing algorithm seems to be better than the LR Cholesky algorithm. If the matrix is small enough to be handled in dense arithmetic (and is not an $\mathcal{H}_\ell(1)$-matrix), then dense eigensolvers, like the LAPACK function `dsyev`, are superior.

12. The $\mathcal{H}$-PINVIT and the LDL^T slicing algorithm require only an almost linear amount of storage. They can handle larger matrices than eigenvalue algorithms for dense matrices.

13. For $\mathcal{H}_\ell$-matrices of local rank 1, the LDL^T slicing algorithm and the LR Cholesky algorithm need almost the same time for the computation of all eigenvalues. For large matrices, both algorithms are faster than the dense LAPACK function `dsyev`.

BIBLIOGRAPHY

[1] P.-A. Absil, R. Mahony, R. Sepulchre, and P. Van Dooren, *A Grassmann-Rayleigh quotient iteration for computing invariant subspaces*, SIAM Rev., 44 (2002), pp. 57–73. 112

[2] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*, SIAM, Philadelphia, PA, third ed., 1999. xvii, 56, 84, 97, 124

[3] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, editors, *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*, SIAM, Philadelphia, PA, 2000. 8, 13

[4] U. Baur, *Control-Oriented Model Reduction for Parabolic Systems*, Dissertation, Inst. f. Mathematik, TU Berlin, `http://opus.kobv.de/tuberlin/volltexte/2008/1760/`, Jan. 2008. 20, 28

[5] M. Bebendorf, *Approximation of boundary element matrices*, Numer. Math., 86 (2000), pp. 565–589. 10.1007/PL00005410. 24

[6] ——, *Hierarchical Matrices: A Means to Efficiently Solve Elliptic Boundary Value Problems*, vol. 63 of Lecture Notes in Computational Science and Engineering (LNCSE), Springer Verlag, Berlin Heidelberg, 2008. 3, 28, 39, 50, 52

[7] M. Bebendorf and W. Hackbusch, *Stabilized rounded addition of hierarchical matrices*, Numer. Lin. Alg. Appl., 14 (2007), pp. 407–423. 136

[8] P. Benner and H. Fassbender, *The symplectic eigenvalue problem, the butterfly form, the SR algorithm, and the Lanczos method*, Linear Algebra Appl., 275/276 (1998), pp. 19–47. 1

[9] ——, *An implicitly restarted symplectic Lanczos method for the symplectic eigenvalue problem*, SIAM J. Matrix Anal. Appl., 22 (2000), pp. 682–713. 1

[10] P. Benner, H. Fassbender, and D. S. Watkins, *Two connections between the*

SR and HR eigenvalue algorithms, Linear Algebra Appl., 272 (1997), pp. 17–32. 71

[11] P. Benner and T. Mach, *On the QR decomposition of $\mathcal{H}$-matrices*, Computing, 88 (2010), pp. 111–129. xxi, 50

[12] ——, *Computing all or some eigenvalues of symmetric $\mathcal{H}_\ell$-matrices*, SIAM J. Sci. Comput., 34 (2012), pp. A485–A496. xxi

[13] ——, *The LR Cholesky algorithm for symmetric hierarchical matrices*, Max Planck Institute Magdeburg Preprint MPIMD/12-05, February 2012. Submitted. xxi

[14] ——, *The preconditioned inverse iteration for hierarchical matrices*, Numer. Lin. Alg. Appl., (2012). 17 pages. xxi

[15] C. H. Bischof and G. Quintana-Ortí, *Algorithm 782: Codes for rank-revealing QR factorizations of dense matrices.*, ACM Trans. Math. Software, 24 (1998), pp. 254–257. 7

[16] S. Börm, *Data-sparse approximation of non-local operators by $\mathcal{H}^2$-matrices*, Linear Algebra Appl., 422 (2007), pp. 380–403. 111

[17] ——, *Approximation of solution operators of elliptic partial differential equations by $\mathcal{H}$- and $\mathcal{H}^2$-matrices*, Numer. Math., 115 (2010), pp. 165–193. 115, 116

[18] ——, *Efficient Numerical Methods for Non-local Operators*, no. 14 in EMS Tracts in Mathematics, Europ. Math. Soc., Zürich, 2010. 19, 21, 40, 111

[19] S. Börm and J. Gördes, *An exact solver for simple H-matrix systems*, preprint, Christian Albrechts Univsität zu Kiel, February 2010. 45

[20] S. Börm and L. Grasedyck, *Hybrid cross approximation of integral operators*, Numer. Math., 101 (2005), pp. 221–249. 24, 111

[21] S. Börm, L. Grasedyck, and W. Hackbusch, *Introduction to hierarchical matrices with applications*, Engineering Analysis with Boundary Elements, 27 (2003), pp. 405–422. 28, 111

[22] K. Braman, R. Byers, and R. Mathias, *The multi-shift QR-algorithm: Aggressive early deflation*, SIAM J. Matrix Anal. Appl., 23 (2002), pp. 948–989. 13

[23] C. L. Bris, *Computational chemistry from the perspective of numerical analysis*, Acta Numer., 14 (2005), pp. 363–444. 129

[24] J. R. Bunch and L. Kaufman, *Some stable methods for calculating inertia and solving symmetric linear systems*, Math. Comp., 31 (1977), pp. 163–179. 92

[25] T. Chan, *Rank revealing QR factorizations*, Linear Algebra Appl., 88/89 (1987), pp. 67–82. 7

[26] S. Chandrasekaran and M. Gu, *A fast and stable solver for recursively semi-separable systems of linear equations*, in Structured Matrices in Mathematics, Com-

puter Science, and Engineering: Proceedings of an AMS-IMS-SIAM Joint Summer Research Conference, University of Colorado, Boulder, June 27-July 1, 1999, American Mathematical Society, 2001. 42

[27] ——, *A divide-and-conquer algorithm for the eigendecomposition of symmetric block-diagonal plus semiseparable matrices*, Numer. Math., 96 (2004), pp. 723–731. 42, 45

[28] S. Chandrasekaran, M. Gu, X. S. Li, and J. Xia, *Fast algorithms for hierachically semiseparable matrices*, preprint to [109], 2007. 42

[29] S. Chandrasekaran, M. Gu, and W. Lyons, *A fast adaptive solver for hierarchically semiseparable representations*, Calcolo, 42 (2005), pp. 171–185. 42

[30] S. Chandrasekaran, M. Gu, and T. Pals, *A fast ULV decomposition solver for hierarchically semiseparable representation*, SIAM J. Matrix Anal. Appl., 28 (2006), pp. 603–622. 42

[31] D. Coppersmith and S. Winograd, *Matrix multiplication via arithmetic progressions*, J. Symbolic Comput., 9 (1990), pp. 251–280. 11

[32] J. J. M. Cuppen, *A divide and conquer method for the symmetric tridiagonal eigenproblem*, Numer. Math., 36 (1981), pp. 177–195. 13, 45

[33] T. Davis and Y. Hu, *University of Florida Sparse Matrix Collection.* `http://www.cise.ufl.edu/research/sparse/matrices/`, 2010. 38

[34] S. Delvaux, K. Frederix, and M. Van Barel, *Transforming a hierarchical into a unitary-weight representation*, Electr. Trans. Num. Anal., 33 (2009), pp. 163–188. 2, 46

[35] S. Delvaux and M. Van Barel, *Structures preserved by the QR-algorithm*, J. Comput. Appl. Math., 187 (2005), pp. 29–40. 71, 79

[36] ——, *Rank structures preserved by the QR-algorithm: the singular case*, J. Comput. Appl. Math., 189 (2006), pp. 157–178. 71

[37] J. W. Demmel, *Applied Numerical Linear Algebra*, SIAM, Philadelphia, PA, 1997. 12

[38] P. Dewilde and S. Chandrasekaran, *A hierarchical semi-separable Moore-Penrose equation solver*, in Wavelets, Multiscale Systems and Hypercomplex Analysis, vol. 167 of Oper. Theory Adv. Appl., Springer, 2006, pp. 69–85. 42

[39] D. Fasino, *Rational Krylow matrices and QR steps on Hermitian diagonal-plus-semiseparable matrices*, Numer. Lin. Alg. Appl., 12 (2005), pp. 743–754. 1, 46, 71, 75

[40] K. V. Fernando and B. N. Parlett, *Accurate singular values and differential qd algorithms*, Numer. Math., 67 (1994), pp. 191–229. 13, 32

[41] J. G. F. Francis, *The QR transformation a unitary analogue to the LR transformation – part 1*, Comput. J., 4 (1961), pp. 265–271. 13, 70

[42] ——, *The QR transformation – part 2*, Comput. J., 4 (1962), pp. 332–345. 13, 71

[43] J. R. Gilbert, C. Moler, and R. Schreiber, *Sparse matrices in MATLAB: Design and implementation*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 333–356. 11

[44] G. H. Golub and W. Kahan, *Calculating the singular values and pseudo inverse of a matrix*, SIAM J. Number. Anal., (1965), pp. 205–224. 13

[45] G. H. Golub and H. A. Van der Vorst, *Eigenvalue computation in the 20th century*, J. Comput. Appl. Math., 123 (2000), pp. 35–65. 1, 2, 8

[46] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, third ed., 1996. 7, 9, 12, 13, 14, 133

[47] J. Gördes, *Eigenwertproblem von hierarchischen Matrizen mit lokalem Rang 1*, Diplomarbeit, Mathematisch-Naturwissenschaftlichen Fakultät der Christian-Albrechts-Universität zu Kiel, May 2009. 2, 30, 45

[48] L. Grasedyck, *Theorie und Anwendungen Hierarchischer Matrizen*, Dissertation, University of Kiel, Kiel, Germany, 2001. available at `http://e-diss.uni-kiel.de/diss_454`. xv, 3, 19, 21, 26, 28, 29, 111, 115, 117

[49] L. Grasedyck and W. Hackbusch, *Construction and arithmetics of $\mathcal{H}$-matrices*, Computing, 70 (2003), pp. 295–334. 19, 21, 22, 23, 24, 25, 27, 28

[50] L. Grasedyck, W. Hackbusch, and B. N. Khoromskij, *Solution of large scale algebraic matrix Riccati equations by use of hierarchical matrices*, Computing, 70 (2003), pp. 121–165. 111

[51] L. Grasedyck, R. Kriemann, and S. Le Borne, *Domain decomposition based $\mathcal{H}$-LU preconditioning*, Numer. Math., 112 (2009), pp. 565–600. 36

[52] L. Greengard and V. Rokhlin, *A fast algorithm for particle simulations*, J. Comput. Phy., 73 (1987), pp. 325–348. 17

[53] M. H. Gutknecht and B. N. Parlett, *From qd to LR, or, how were the qd and LR algorithms discovered?*, IMA J. Numer. Anal., 31 (2010), pp. 741–754. 70

[54] W. Hackbusch, *Integral equations: theory and numerical treatment*, International series of numerical mathematics, Birkhäuser, 1995. 14, 15, 16

[55] ——, *A Sparse Matrix Arithmetic Based on $\mathcal{H}$-Matrices. Part I: Introduction to $\mathcal{H}$-Matrices*, Computing, 62 (1999), pp. 89–108. 17, 28, 30, 90

[56] ——, *Hierarchische Matrizen. Algorithmen und Analysis*, Springer-Verlag, Berlin, 2009. xv, 6, 15, 19, 26, 27, 28, 29, 30, 116

[57] W. Hackbusch and M. Bebendorf, *Existence of $\mathcal{H}$-Matrix Approximants to the Inverse FE-Matrix of elliptic Operators with L^∞-Coefficients*, Numer. Math., 95 (2003), pp. 1–28. 115

[58] W. Hackbusch, B. N. Khoromskij, and R. Kriemann, *Hierarchical matrices based on a weak admissibility criterion*, Computing, 73 (2004), pp. 207–243. 21, 28, 32, 33, 82, 84, 95

[59] W. Hackbusch, B. N. Khoromskij, S. Sauter, and E. Tyrtyshnikov, *Use of tensor formats in elliptic eigenvalue problems*, MPI MiS Leipzig Preprint 78/2008, 2008. 130

[60] W. Hackbusch and W. Kress, *A projection method for the computation of inner eigenvalues using high degree rational operators*, Computing, 81 (2007), pp. 259–268. 2, 44, 138

[61] W. Hackbusch and Z. P. Nowak, *On the fast matrix multiplication in the boundary element method by panel clustering*, Numer. Math., 54 (1989), pp. 463–491. 10.1007/BF01396324. 17

[62] G. Henry and R. Van de Geijn, *Parallelizing the QR algorithm for the unsymmetric algebraic eigenvalue problem: myths and reality*, SIAM J. Sci. Comput., 17 (1996), pp. 870–883. 105

[63] N. J. Higham, *Computing the polar decomposition with applications*, SIAM J. Sci. Stat. Computing, 7 (1986), pp. 1160–1174. xv, 51

[64] ——, *Accuracy and Stability of Numerical Algorithms*, SIAM, Philadelphia, PA, 2002. 12, 13, 52

[65] *$\mathcal{H}$Lib 1.3/1.4*. `http://www.hlib.org`, 1999-2012. xvii, 29, 34, 36, 39, 47, 51, 73, 97

[66] I. C. F. Ipsen, *A history of inverse iteration*, in Helmut Wielandt: Mathematische Werke, Mathematical Works, B. Huppert and H. Schneider, eds., vol. 2, Walter de Gruyter, 1996, pp. 464–472. 111

[67] B. N. Khoromskij, V. Khoromskaia, and H. Flad, *Numerical solution of the Hartree-Fock equation in multilevel tensor-structured format*, SIAM J. Sci. Comput., 33 (2011). 130

[68] A. V. Knyazev, *Preconditioned eigensolvers — an oxymoron?*, Electr. Trans. Num. Anal., 7 (1998), pp. 104–123. 3, 113

[69] ——, *Toward the optimal preconditioned eigensolver: locally optimal block preconditioned conjugate gradient method*, SIAM J. Sci. Comput., 23 (2001), pp. 517–541. 114

[70] A. V. Knyazev and K. Neymeyr, *A geometric theory for preconditioned inverse iteration III: A short and sharp convergence estimate for generalized eigenvalue*

problems, Linear Algebra Appl., 358 (2003), pp. 95–114. 113

[71] ——, *Gradient flow approach to geometric convergence analysis of preconditioned eigensolvers*, SIAM J. Matrix Anal. Appl., 31 (2009), pp. 621–628. 113, 114

[72] J. Koch, W. Hackbusch, and K. Sundmacher, *$\mathcal{H}$-matrix methods for linear and quasi-linear integral operators appearing in population balances*, Comp. & Chem. Eng., 31 (2007), pp. 745–759. 14

[73] M. Lintner, *Lösung der 2D Wellengleichung mittels hierarchischer Matrizen*, Dissertation, Fakultät für Mathematik, TU München, `http://tumb1.biblio.tu-muenchen.de/publ/diss/ma/2002/lintner.pdf`, Jun. 2002. 3, 28, 50, 51, 92, 116

[74] ——, *The Eigenvalue Problem for the 2D Laplacian in $\mathcal{H}$-Matrix Arithmetic and Application to the Heat and Wave Equation*, Computing, 72 (2004), pp. 293–323. 51, 111

[75] T. Mach, *Lösung von Randintegralgleichungen zur Bestimmung der Kapazitätsmatrix von Elektrodenanordnungen mittels $\mathcal{H}$-Arithmetik*, Diplomarbeit, TU Chemnitz, May 2008. 17

[76] ——, *Computing inner eigenvalues of matrices in tensor train matrix format*, in ENUMATH 2011 Proceedings Volume, 2011. Accepted, available as Max Planck Institute Magdeburg Preprint MPIMD/11-09, 11 pages. 123, 130

[77] N. Mastronardi, S. Chandrasekaran, and S. van Huffel, *Fast and stable reduction of diagonal plus semi-separable matrices to tridiagonal and bidiagonal form*, BIT, 41 (2001), pp. 149–157. 10.1023/A:1021973919347. 42

[78] N. Mastronardi, E. Van Camp, and M. Van Barel, *Divide and conquer algorithms for computing the eigendecomposition of symmetric diagonal-plus-semiseparable matrices*, Numer. Algorithms, 39 (2005), pp. 379–398. 1, 13, 42

[79] R. B. Morgan, *Computing interior eigenvalues of large matrices*, Linear Algebra Appl., 154–156 (1991), pp. 289–309. 120, 121

[80] K. Neymeyr, *A geometric theory for preconditioned inverse iteration I: Extrema of the Rayleigh quotient*, Linear Algebra Appl., 322 (2001), pp. 61–85. 111, 113, 117

[81] ——, *A geometric theory for preconditioned inverse iteration II: Convergence estimates*, Linear Algebra Appl., 322 (2001), pp. 87–104. 113

[82] ——, *A Hierarchy of Preconditioned Eigensolvers for Elliptic Differential Operators*, Habilitationsschrift, Mathematische Fakultät der Universität Tübingen, Sep. 2001. 113, 118

[83] ——, *A geometric theory for preconditioned inverse iteration applied to a subspace*, Math. Comp., 71 (2002), pp. 197–216. 117, 118

[84] *OpenMP*. http://openmp.org, 1997–2012. xvii, 105

[85] I. V. Oseledets and E. E. Tyrtyshnikov, *Breaking the curse of dimensionality or how to use SVD in many dimensions*, SIAM J. Sci. Comput., 31 (2009), pp. 3744–3759. 130

[86] L. Page, S. Brin, R. Motwani, and T. Winograd, *The PageRank citation ranking: Bringing order to the web*, tech. rep., Stanford InfoLab, 1998. 110

[87] B. N. Parlett, *The Symmetric Eigenvalue Problem*, Prentice-Hall, Englewood Cliffs, first ed., 1980. 2, 3, 8, 9, 89, 91

[88] B. Plestenjak, M. Van Barel, and E. Van Camp, *A Cholesky LR algorithm for the positive definite symmetric diagonal-plus-semiseparable eigenproblem*, Linear Algebra Appl., 428 (2008), pp. 586–599. 1, 2, 3, 13, 42, 46, 75, 76

[89] H. Rutishauser, *Bestimmung der Eigenwerte und Eigenvektoren einer Matrix mit Hilfe des Quotienten-Differenzen-Algorithmus*, Z. Angew. Math. Phys., 6 (1955), pp. 387–401. 13, 70

[90] ——, *Solution of eigenvalue problems with the LR-transformation*, Nat. Bur. Standards Appl. Math. Ser., 49 (1958), pp. 47–81. 70

[91] ——, *Über eine kubisch konvergente Variante der LR-Transformation*, ZAMM Z. Angew. Math. Mech., 40 (1960), pp. 49–54. 70

[92] Y. Saad, *Iterative Methods for Sparse Linear Systems*, SIAM, Philadelphia, PA, 2003. 10

[93] S. Sauter and C. Schwab, *Randelementmethoden: Analyse, Numerik und Implementierung schneller Algorithmen*, B.G. Teubner, Wiesbaden, 2004. 16

[94] G. Schulz, *Iterative berechung der reziproken matrix*, ZAMM Z. Angew. Math. Mech., 13 (1933), pp. 57–59. 28

[95] G. W. Stewart, *Four algorithms for the efficient computation of truncated pivoted QR approximations to a sparse matrix*, Numer. Math., 83 (1999), pp. 313–323. 61

[96] G. Strang and T. Nguyen, *The interplay of ranks of submatrices*, SIAM Rev., 46 (2004), pp. 637–646. 42

[97] The Open MPI Project, *Open MPI*. http://open-mpi.org, 2004–2012. xvii, 105

[98] E. E. Tyrtyshnikov, *Mosaic-skeleton approximations*, Calcolo, 33 (1996), pp. 47–57. 17

[99] ——, *Incomplete cross approximation in the mosaic-skeleton method*, Computing, 64 (2000), pp. 367–380. 10.1007/s006070070031. 24

[100] R. Vandebril, M. Van Barel, G. H. Golub, and N. Mastronardi, *A*

bibliography on semiseparable matrices, Calcolo, 42 (2005), pp. 249–270. 40

[101] R. Vandebril, M. Van Barel, and N. Mastronardi, *An implicit Q theorem for Hessenberg-like matrices*, Mediterranean J. Math., 2 (2005), pp. 259–275. 71

[102] ——, *An implicit QR algorithm for symmetric semiseparable matrices*, Numer. Lin. Alg. Appl., 12 (2005), pp. 625–658. 13, 42, 71

[103] ——, *Matrix Computations and Semiseparable Matrices. Vol. I+II*, Johns Hopkins University Press, Baltimore, MD, 2008. 13, 41

[104] L.-W. Wang and A. Zunger, *Electronic structure pseudopotential calculations of large (∼1000 atoms) Si quantum dots*, J. Phys. Chem., 98 (1994), pp. 2158–2165. 3, 112, 113, 120

[105] D. S. Watkins, *QR-like algorithms for eigenvalue problems*, J. Comput. Appl. Math., 123 (2000), pp. 67–83. 71

[106] D. S. Watkins and L. Elsner, *Convergence of algorithms of decomposition type for the eigenvalue problem*, Linear Algebra Appl., 143 (1995). 71

[107] J. H. Wilkinson, *The Algebraic Eigenvalue Problem*, Oxford University Press, Oxford, 1965. 7, 8, 10, 70, 79, 87, 109, 120

[108] P. Willems, *On MR^3-type Algorithms for the Tridiagonal Symmetric Eigenproblem and the Bidiagonal SVD*, Dissertation, Bergischen Universität Wuppertal, 2010. 13, 32

[109] J. Xia, S. Chandrasekaran, M. Gu, and X. S. Li, *Fast algorithms for hierachically semiseparable matrices*, Numer. Lin. Alg. Appl., 17 (2010), pp. 953–976. 42, 83, 103, 108, 147

[110] H. Xu, *The relation between the QR and LR algorithms*, SIAM J. Matrix Anal. Appl., 19 (1998), pp. 551–555. 71

INDEX