Fast hierarchical solvers for symmetric eigenvalue problems

THÈSE N^{O} 8808 (2018)

PRÉSENTÉE LE 24 AOÛT 2018

À LA FACULTÉ DES SCIENCES DE BASE
ALGORITHMES NUMÉRIQUES ET CALCUL HAUTE PERFORMANCE - CHAIRE CADMOS
PROGRAMME DOCTORAL EN MATHÉMATIQUES

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Ana ŠUŠNJARA

acceptée sur proposition du jury:

Prof. T. Mountford, président du jury
Prof. D. Kressner, directeur de thèse
Prof. Y. Nakatsukasa, rapporteur
Prof. R. Vandebril, rapporteur
Prof. J. Hesthaven , rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE
Suisse
2018

Acknowledgements

First and foremost, I would like to thank my advisor Prof. Daniel Kressner for giving me the opportunity to work in his research group, for his guidance during my PhD studies, and for pushing me forward. His critical thinking, sharpness and his immense knowledge have been a constant source of inspiration for me.

Besides my advisor, I would like to express my gratitude to Prof. Jan Hesthaven, Prof. Yuji Nakatsukasa and Prof. Raf Vandebril who kindly accepted to be a part of my jury and to Prof. Thomas Mountford for presiding it.

A big thank you goes to my ANCHP mates: Agnieszka, André, Francesco, Francisco, Jonas, Meiyue, Michael, Petar, Robert, Stefano and Zvonimir, for all the help, support and laughs during the last years, which made my stay at EPFL even nicer. I would like to acknowledge Jonas for his help with hierarchical matrices, and Francesco and Stefano for being an amazing support in the last period of my PhD studies. I am also thankful to all the colleagues in MATHICSE for making a very friendly working environment, and for various discussions during both work and non-work related events.

My time at EPFL was made enjoyable in large part due to the many new friends that became a part of my life. I am grateful for the time spent with Enrica and Lucien, who were wonderful flatmates, and created a very supportive and homely atmosphere. To Enrica, thank you for being like a sister to me. I would also like to express my appreciation to the friends I met in Lausanne, who made the PhD experience very interesting, exciting and full of changes.

Moreover, I would like to thank all my friends from Croatia for all the great time that we have shared and their encouragement in times of need. In particular, I wish to thank Katarina, Mario and Sara for their moral support and making my days more cheerful, even though they were far away.

Furthermore, I am deeply thankful to Thierry for his love, support, and limitless patience. A huge thank you goes to Petar, for being my closest companion on this journey, and for being there for me both in beautiful and difficult moments of my PhD.

Acknowledgements

Lastly, I would like to express my gratitude to my family. Their unconditional love, support and belief in my success kept me going regardless of the challenge that I faced.

I dedicate this thesis to the memory of my grandfather Anđelko, whose role in my life was, and remains, immense.

Lausanne, August 2018 Ana

Abstract

In this thesis we address the computation of a spectral decomposition for symmetric banded matrices. In light of dealing with large-scale matrices, where classical dense linear algebra routines are not applicable, it is essential to design alternative techniques that take advantage of data properties. Our approach is based upon exploiting the underlying hierarchical low-rank structure in the intermediate results. Indeed, we study the computation in the *hierarchically off-diagonal low rank* (HODLR) format of two crucial tools: QR decomposition and spectral projectors, in order to devise a fast spectral divide-and-conquer method.

In the first part we propose a new method for computing a QR decomposition of a HODLR matrix, where the factor R is returned in the HODLR format, while Q is given in a compact WY representation. The new algorithm enjoys linear-polylogarithmic complexity and is norm-wise accurate. Moreover, it maintains the orthogonality of the Q factor.

The approximate computation of spectral projectors is addressed in the second part. This problem has raised some interest in the context of linear scaling electronic structure methods. There the presence of small spectral gaps brings difficulties to existing algorithms based on approximate sparsity. We propose a fast method based on a variant of the QDWH algorithm, and exploit that QDWH applied to a banded input generates a sequence of matrices that can be efficiently represented in the HODLR format. From the theoretical side, we provide an analysis of the structure preservation in the final outcome. More specifically, we derive a priori decay bounds on the singular values in the off-diagonal blocks of spectral projectors. Consequently, this shows that our method, based on data-sparsity, brings benefits in terms of memory requirements in comparison to approximate sparsity approaches, because of its logarithmic dependence on the spectral gap. Numerical experiments conducted on tridiagonal and banded matrices demonstrate that the proposed algorithm is robust with respect to the spectral gap and exhibits

linear-polylogarithmic complexity. Furthermore, it renders very accurate approximations to the spectral projectors even for very large matrices.

The last part of this thesis is concerned with developing a fast spectral divide-and-conquer method in the HODLR format. The idea behind this technique is to recursively split the spectrum, using invariant subspaces associated with its subsets. This allows to obtain a complete spectral decomposition by solving the smaller-sized problems. Following Nakatsukasa and Higham, we combine our method for the fast computation of spectral projectors with a novel technique for finding a basis for the range of such a HODLR matrix. The latter strongly relies on properties of spectral projectors, and it is analyzed theoretically. Numerical results confirm that the method is applicable for large-scale matrices, and exhibits linear-polylogarithmic complexity.

Keywords. Symmetric eigenvalue problem, fast spectral divide-and-conquer, spectral projectors, spectral gap, large-scale problems, hierarchical matrices, HODLR format, low-rank structure, QR decomposition

Estratto

L'oggetto di questa tesi è il calcolo della decomposizione spettrale di matrici simmetriche a banda. Nell'ottica di affrontare problemi a larga scala, in cui gli algoritmi classici di algebra lineare densa non sono applicabili, è essenziale progettare tecniche alternative che si avvantaggino di particolari proprietá dei dati. Il nostro approccio si basa sullo sfruttare la struttura gerarchica di rango basso presente nei risultati intermedi. Infatti, studiamo il calcolo nel formato HODLR di due strumenti cruciali: la fattorizzazione QR e i proiettori spettrali, al fine di ottenere un metodo spettrale divide et impera con complessitá subquadratica.

Nella prima parte, proponiamo un nuovo algoritmo per il calcolo della fattorizzazione QR di una matrice HODLR che restituisce il fattore R nel formato HODLR e la matrice Q nella sua rappresentazione WY compatta. Questa procedura ha complessitá lineare polilogaritmica ed è accurata in norma. Inoltre, riesce a mantenere l'ortogonalitá del fattore Q.

Il calcolo approssimato dei proiettori spettrali è trattato nella seconda parte. Questo problema ha recentemente destato interesse nel contesto di metodi numerici per la struttura elettronica. In questa applicazione, la prossimitá di due o piú autovalori (gap spettrale) causa difficoltá agli algoritmi esistenti, che si basano su approsimare la sparsitá dei risultati intermedi. Proponiamo una variante del metodo QDWH che sfrutta il fatto che il metodo QDWH, applicato ad una matrice a banda, genera una successione di matrici che possono essere approssimate efficientemente nel formato HODLR. Dal punto di vista teorico, forniamo un'analisi della preservazione della struttura nel risultato finale. In particolare, deriviamo stime a priori del decadimento dei valori singolari nei blocchi fuori dalla diagonale dei proiettori spettrali. Queste stime evidenziano una dipendenza del rate di decadimento dal logaritmo del gap spettrale. Di conseguenza si giustifica il vantaggio, in termini di utilizzo di memoria, di sfruttare la data-sparsitá rispetto agli approcci basati sulla sparsitá. Gli esperimenti numerici effettuati su matrici tridiagonali

e a banda dimostrano che l'algoritmo proposto è robusto rispetto al gap spettrale ed ha complessitá lineare polilogaritmica. Inoltre, riesce a fornire approssimazioni accurate anche per matrici di dimensioni considerevoli.

L'ultima parte di questa tesi è dedicata allo sviluppo di un metodo spettrale divide et impera per matrici nel formato HODLR. L'idea alla base di questa tecnica è di dividere ricorsivamente lo spettro e calcolare i corrispondenti sottoinsiemi degli autovalori utilizzando appropriati sottospazi invarianti. Questo consente di ottenere la fattorizzazione spettrale completa risolvendo problemi analoghi di dimensioni piú piccola. Seguendo lo schema di Nakatsukasa e Higham, combiniamo il nostro metodo per il calcolo veloce dei proiettori spettrali con un nuovo metodo per estrarre una base dell'immagine di una matrice nel formato HODLR. Le garanzie di funzionamento di quest'ultimo sono garantite dalle proprietá dei proiettori spettrali, che vengono analizzate teoricamente. I risultati numerici confermano la scalabilitá del metodo a matrici di grandi dimensioni, essendo la sua complessitá lineare polilogaritmica.

Parole chiave. Problema agli autovalori simmetrico, metodo veloce divide et impera spettrale, proiettore spettrale, gap spettrale, problemi a larga scala, matrici gerarchiche, formato HODLR, struttura di rango basso, fattorizzazione QR

Contents

1 Introduction and background

This thesis is concerned with the computation of a complete spectral decomposition of a large-scale n-by-n symmetric banded matrix. In particular, we develop a new fast spectral divide-and-conquer method for the computation of all eigenvalues and eigenvectors, with the eigenvectors given in a structured form. Our approach is based on the fast computation of spectral projectors associated with roughly half of the spectrum, and the extraction of the related invariant subspaces. The underlying structure of spectral projectors of banded matrices motivates the use of data-sparse formats and the related approximate arithmetics, which results in methods with linear-polylogarithmic complexity. More specifically, in this thesis we focus on the HODLR format.

The thesis is organized as follows. In Chapter 2 we review the HODLR format and the related matrix formats, alongside the HODLR arithmetics. We develop a new fast method for the computation of a QR decomposition of a HODLR matrix. Chapter 3 is an overview chapter, which discusses several existing eigenvalue methods for hierarchical matrices. Chapter 4 is concerned with the fast computation of spectral projectors of banded matrices. Lastly, in Chapter 5 we present a new fast spectral divide-and-conquer method for banded matrices.

In the rest of this chapter we first provide a concise introduction to the main ingredients of this thesis: the symmetric eigenvalue problem and data-sparse formats, followed by an overview of important definitions. The chapter is concluded with the thesis' contributions.

1.1 Symmetric eigenvalue problem

For a symmetric matrix $A \in \mathbb{R}^{n\times n}$, a pair (λ, q) satisfying

$$Aq = \lambda q, \quad \lambda \in \mathbb{R}, q \in \mathbb{R}^n, q \neq 0,$$

is called an *eigenpair* of A. The scalar λ is called an *eigenvalue* of A, and the vector q is a corresponding *eigenvector*. The set of all eigenvalues $\{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ of A is called the *spectrum* of A, and is denoted by $\sigma(A)$. Matrix A admits a *spectral decomposition*

$$A = Q\Lambda Q^T, \tag{1.1}$$

with $\Lambda = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$ and $Q \in \mathbb{R}^{n\times n}$ orthogonal.

The symmetric eigenvalue problem is one of the most studied problems in the field of numerical linear algebra, with wide applications in science and engineering, varying from quantum mechanics, structural engineering, through graph theory, Markov chains to data and market analysis, and many more.

Depending on a specific application, it is required to compute the extremal eigenpairs, a subset of eigenpairs, or a complete spectral decomposition. In the last decades, a number of methods have been developed to tackle this problem.

Most of the approaches for solving a symmetric eigenvalue problem proposed in the literature consists of two steps: reducing a dense symmetric matrix to tridiagonal form, which is followed by the computation of the spectral decomposition of a tridiagonal matrix. The second step is performed with a combination of standard methods, such as the QR algorithm [52, 53, 82, 94], the classical divide-and-conquer (D&C) method [36, 64] or the algorithm of multiple relatively robust representations (MRRR) [43, 44]. These methods exhibit $\mathcal{O}(n^2)$ complexity or higher, since all eigenvectors are computed and stored explicitly. When interested in a large scale eigenvalue problem, there is therefore a necessity to exploit a hidden structure in the data, in addition to *sparsity*, in order to reduce both the computational and the storage complexity.

A growing line of research takes advantage of the underlying matrix structure to derive fast methods for the symmetric eigenvalue problem. This involves eigenvalue methods for hierarchical matrices and $\mathcal{H}^2$–matrices [16, 18, 61, 70], quasiseparable matrices [48],

(hierarchically) semiseparable matrices [110, 111, 114], and for rank structured matrices given in terms of Givens-weight representations [39, 40], such as the QR algorithm, bisection method, preconditioned inverse iteration, and the divide-and-conquer method. Several of these approaches involving hierarchical matrices are reviewed in Chapter 3.

1.2 Data-sparse formats

The simplest form of structured matrices are *sparse matrices*. We say that an $n \times n$ matrix is sparse if it can be represented with only $\mathcal{O}(n)$ parameters. An important class of sparse matrices are banded matrices, arising for instance in the discretization of 1D partial differential equations. Fast methods based on *approximate sparsity*, such as the computation of matrix functions or fast linear solvers, have already been addressed in literature; see e.g. [21, 22, 23, 59, 99]. In particular, for a banded A and a function f that can be well approximated by a low-degree polynomial on the spectrum of A, $f(A)$ admits a good approximation by a sparse matrix. Nevertheless, this approach is limited to a certain class of functions, and performs badly for functions with singularities close to the spectrum. However, if instead of polynomials one considers a larger family of approximants, rational functions, it might be beneficial to turn to more general matrix structures as well, to the so called *data-sparse* formats. Specifically, a $n \times n$ matrix is said to be *data-sparse* if it allows a representation using much less than n^2 parameters.

An advantage of using a data-sparse format is easily noticed for the inverse of a tridiagonal matrix. Consider

$$A = (256+1)^2 \begin{bmatrix} 2 & -1 & & & & & \\ -1 & 2 & -1 & & & & \\ & \ddots & \ddots & \ddots & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & -1 & 2 & -1 & \\ & & & & -1 & 2 & \end{bmatrix} \in \mathbb{R}^{256\times 256},$$

and its inverse. All elements in A^{-1} are larger than machine precision, which implies that the sparsity is lost. In theory, a result by Demko et al. [41] gives a justification to approximate A^{-1} by a banded matrix. However, Figure 1.1 reveals that the decay in the off-diagonal elements is very slow, and therefore the approach based on the approximate

sparsity does not seem to be adequate for this case.

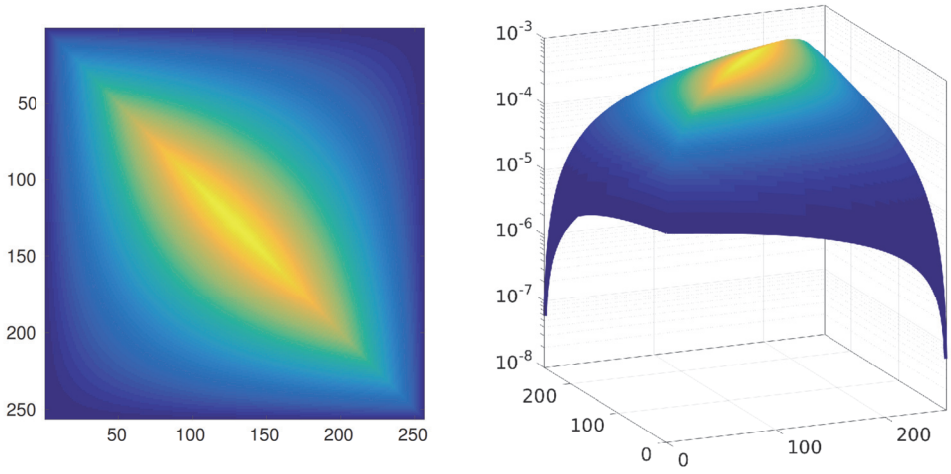

Figure 1.1 – The intensity plot in 2D (left) and 3D (right) of the elements of the inverse of a tridiagonal matrix A with $\kappa(A) = 2.7 \cdot 10^4$.

On the other hand, the inverse of a triangular matrix exhibits an important property - all off-diagonal submatrices in A^{-1} are of rank one [109, Chapter 1]. Hence by subdividing the diagonal blocks of A^{-1} recursively into four submatrices and representing the off-diagonal blocks via their rank-1 factors, as explained in Chapter 2, the memory needed to exactly store A^{-1} can be reduced to $\mathcal{O}(n \log n)$.

In the 1990s Hackbusch [66] introduced a concept of *hierarchical matrices* in the context of integral equations. Hierarchical matrices are a data-sparse format built upon the fact a matrix allows a decomposition into submatrices, where suitable submatrices can be well approximated by a low-rank matrix. This in turn results in the reduction of storage demands to $\mathcal{O}(n \log n)$. Depending on the matrix subdivision and the presence of additional structure properties, e.g. nestedness condition, we obtain different possible hierarchical matrices. This includes HODLR matrices [67, 69], $\mathcal{H}^2$–matrices [29] and hierarchically semiseparable (HSS) matrices [32, 115]. We note that another line of research concerning rank structured matrices also includes the efficient handling of semiseparable matrices [109, 110], and quasiseparable matrices [45, 46, 47, 48].

The arithmetics with hierarchical matrices is approximative, involving a compression in the low-rank blocks, with respect to a given accuracy or a prescribed rank, in order to keep ranks reasonably low. Most of the dense linear algebra operations, such as

matrix-vector products, matrix-matrix products, matrix decompositions, solving linear equations, etc., can be performed with linear-polylogarithmic cost, $\mathcal{O}(n \log^{\alpha} n)$, with $\alpha \geq 1$. Almost linear storage requirements and fast arithmetics allow us to treat a wide variety of large-scale computational problems, especially considering a huge demand for dealing with big data problems in the last years. To name a few, the applications include matrix function computations [9, 55, 56, 62, 68, 80], design of fast linear solvers [3, 32, 33, 57, 92, 97, 113, 115] and many others. A more thorough overview is given in Chapter 2.

1.3 Background

This section is devoted to review some of the basic definitions and matrix decompositions which are heavily used throughout this thesis. For a deeper insight into the topics, we refer the reader to the books [60, 77, 79].

Banded matrix. Let $A \in \mathbb{R}^{n \times n}$ and $b \in \{0, 1, \dots n-1\}$. We say that $A = (a_{ij})$ is a *banded matrix* with bandwidth b if $a_{ij} = 0$ whenever $|i - j| > b$. A tridiagonal matrix is 1-banded matrix, while a diagonal matrix is 0-banded matrix.

Hessenberg matrix. A matrix $A \in \mathbb{R}^{n \times n}, A = (a_{ij})$, is called *upper Hessenberg* if $a_{ij} = 0$ for all i, j such that $i > j + 1$. A is said to be b-upper Hessenberg if $A_{ij} = 0$ whenever $i > j + b$.

Cholesky decomposition. Let $A \in \mathbb{R}^{n \times n}$ be symmetric positive definite matrix. Then there exists a unique upper triangular matrix $R \in \mathbb{R}^{n \times n}$ with positive diagonal elements such that $A = R^T R$.

QR decomposition. Let $A \in \mathbb{R}^{n \times m}$. Then there exist an orthogonal $Q \in \mathbb{R}^{n \times n}$ and an upper trapezoidal $R \in \mathbb{R}^{n \times m}$ such that $A = QR$. If A has a full column rank, then the thin QR decomposition $A = QR$ is unique with an orthonormal matrix $Q \in \mathbb{R}^{n \times m}$ and an upper triangular matrix R with positive diagonal elements. We note that R is a Cholesky factor of $A^T A$.

SVD decomposition. Let $A \in \mathbb{R}^{n\times m}$. Then there exist orthogonal matrices $U \in \mathbb{R}^{n\times n}$ and $V \in \mathbb{R}^{m\times m}$ such that

$$A = U\Sigma V^T, \quad \Sigma = \operatorname{diag}(\sigma_1, \sigma_2, \ldots, \sigma_p), \quad p = \min\{n, m\}.$$

The scalars $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_p \geq 0$ are called the *singular values* of A, while the columns of U and V are the *left and right singular vectors* of A, respectively. The *rank* of A is equal to the number r of positive singular values of A. The 2-norm condition number of A is given as $\kappa(A) = \sigma_1/\sigma_r$.

Polar decomposition. Let $A \in \mathbb{R}^{n\times m}$ with $n \geq m$. Then there exist an orthonormal matrix $U \in \mathbb{R}^{n\times m}$ and a unique symmetric positive semi-definite matrix $H \in \mathbb{R}^{m\times m}$ such that

$$A = UH, \quad U^TU = I.$$

If A has a full column rank, then H is positive definite and U is unique as well.

Matrix functions. Let $A \in \mathbb{R}^{n\times n}$ be symmetric with the spectral decomposition (1.1). We say that a function f is *defined on the spectrum of* A if all values

$$f(\lambda_i), \quad i = 1, \ldots, n$$

exist. For a function f defined on the spectrum of A, we can define $f(A)$ as

$$f(A) := Qf(\Lambda)Q^T = Q\operatorname{diag}(f(\lambda_1), f(\lambda_2), \ldots, f(\lambda_n))Q^T.$$

We note that there exist other equivalent definitions to $f(A)$, given in terms of the Hermite interpolation polynomial, or the Cauchy integral formula; see [77] for more details.

1.4 Contributions of the thesis

Chapter 2. We discuss several structured matrix formats, such as HODLR matrices, HSS matrices, general hierarchical and $\mathcal{H}^2$-matrices. Our focus is in particular on the HODLR format, which can be used to efficiently store spectral projectors of banded matrices. We derive the arithmetic operations within the HODLR format, which will

be used in the subsequent chapters, and show that they can be performed efficiently, exhibiting a linear-polylogarithmic complexity.

In Section 2.2.9 we propose a new fast method for the computation of a QR decomposition of a HODLR matrix. The new method is based on a recursive block column approach, which exploits a compact WY representation to efficiently and accurately compute the orthogonal factor in the HODLR format. The implementation requires some attention, because of a block structure in the HODLR format. Moreover, we examine the performance of our method on several examples, and compare it to an existing method. The experiments indicate that our method yields satisfactory results, both in terms of accuracy and orthogonality.

Furthermore, Section 2.2.11 describes the functionalities of our MATLAB package, which implements all algorithms described in Chapter 2.

Chapter 3. We review state-of-the-art methods for the symmetric eigenvalue problem involving hierarchical matrix formats.

Chapter 4. We consider the approximate computation of spectral projectors for symmetric banded matrices. While this problem has received considerable attention, especially in the context of linear scaling electronic structure methods, the presence of small relative spectral gaps challenges existing methods based on approximate sparsity. In this work, we show how a data-sparse approximation based on hierarchical matrices can be used to overcome this problem.

We propose a new fast method based on a variant [91] of the QR-based dynamically weighted Halley algorithm (QDWH) [89]. Our approach relies on the fact that the HODLR format is well suited for representing the iterates of the QDWH algorithm applied to a banded matrix. In Section 4.2, we study the off-diagonal blocks of spectral projectors. Based on the best rational approximation to the sign function, we derive a priori bounds on the singular values in the off-diagonal blocks of spectral projectors, and prove that the memory needed for storing the approximate spectral projector in the HODLR format depends only logarithmically on the spectral gap. This represents a major improvement over approximate sparsity approach, where the gap enters memory requirements inverse proportionally. The implementation of the QDWH algorithm in the HODLR format

requires some care, in particular, concerning the computation of the first iterate, which is addressed in Section 4.3. One major contribution of Chapter 4 is to provide a justification and develop a fast method for the exact representation of the first iterate in the HODLR format.

Using the arithmetic in the HODLR format, our new method exhibits a linear-polylogarithmic complexity. Numerical experiments demonstrate that the performance of our algorithm is robust with respect to the spectral gap. A preliminary MATLAB implementation becomes faster than `eig` already for matrix sizes of a few thousand. Moreover, the favorable complexity allows to deal with large scale matrices, up to $n = 1\,000\,000$ on a desktop computer.

This chapter is largely based on the published article [81]. The additional content is added in Section 4.1, where we give a deeper insight into the QDWH method. In Section 4.2, in particular Example 4.9, we demonstrate the quality of our a priori singular values' bounds. Moreover, in Section 4.3 we extend the proof of Theorem 4.11 to a more general case, given in Theorem 4.13, which shows a bound for the off-diagonal ranks in the first iterate of the QDWH algorithm for banded matrices.

Chapter 5. We propose a new fast method for computing all eigenvalues and eigenvectors of a symmetric banded matrix. Our approach is based on the spectral divide-and-conquer method from [91], which recursively splits the matrix spectrum using the invariant subspaces obtained from spectral projectors associated with roughly half of the spectrum.

The computation of spectral projectors is carried out in the HODLR format, employing the method developed in Chapter 4. Following the new method for the QR decomposition presented in Chapter 2, we derive a new algorithm for computing the first iterate in the QDWH algorithm for general symmetric HODLR matrices. This allows us to apply the spectral divide-and-conquer method even for matrices with small spectral gaps.

The computation of an invariant subspace requires to determine a basis for the range of a spectral projector. Given that spectral projectors are stored in the HODLR format, this represents a major challenge, and makes the standard pivoting approaches inapplicable to our case. We therefore derive a novel method for extracting a set of linearly independent columns of a spectral projector in the HODLR format, and provide its theoretical analysis.

The new method, presented in Section 5.2, is built upon a block Cholesky decomposition with local pivoting.

The numerical experiments demonstrate that our algorithm exhibits linear-polylogarithmic complexity and allows for conveniently dealing with large-scale matrices.

This chapter mostly follows the submitted preprint [108]. The content of the chapter contains a major change, which concerns the first iteration of the QDWH algorithm. Indeed, the preprint [108] implements the first iterate using the Cholesky decomposition, while in this work the first iterate is performed using a new method for computing a QR decomposition of a $2n \times n$ matrix $[A \ I]^T$, where A is a HODLR matrix. This results in a more accurate and stable method in comparison to [108], in particular for matrices with small spectral gaps.

2 Matrices with hierarchical low-rank structure

In the last decades, hierarchical matrix representations and the associated arithmetics have become a crucial tool when dealing with large-scale matrices coming from applications with an underlying geometry. Hierarchical matrices are data-sparse approximations of a class of dense matrices. The concept of hierarchically low-rank matrices is based on the observation that certain submatrices can be recursively subdivided and that certain sub-blocks can be well represented in terms of their low-rank approximations. For this class of matrices standard linear algebra operations, like matrix-vector products, matrix factorizations, solving linear equations, etc., are typically defined in a block-wise manner and can be performed in almost linear complexity.

Hierarchical matrices, or more specifically HODLR matrices, are essential objects in this work. The aim of this chapter is to introduce all necessary ingredients to work with HODLR matrices. In particular, in Section 2.1, we first review different matrix formats with hierarchical structure: HODLR matrices, $\mathcal{H}$–matrices based on a strong admissibility condition and HSS matrices. Section 2.2 is dedicated to the arithmetics in the HODLR format, which will be heavily used in subsequent chapters. More specifically, we describe arithmetic operations, such as addition, multiplication, Cholesky decomposition, submatrix selections in detail and provide complexity estimates.

Additionally, in Section 2.2.9 we propose a new algorithm for the computation of a QR decomposition in the HODLR format. Our method is performed recursively by block columns, and exploits a compact WY representation to store the Q factor, which yields satisfactory results in terms of orthogonality. Our approach exhibits linear-polylogarithmic

complexity. Finally, the chapter is concluded by Section 2.2.11, where we give an overview of the functionalities of our HODLR MATLAB package.

The material covered in this chapter up to Section 2.2.9 mostly follows [8, 67] and references therein.

2.1 Hierarchical matrix formats

Hierarchical ($\mathcal{H}$–) matrices have been introduced by Hackbusch [66] in the context of solving boundary integral equations, and since then have been used in [10, 11, 12, 14, 29, 35, 58, 63, 67, 87] and other. $\mathcal{H}$–matrix structures are based on a hierarchical matrix partitioning into sub-blocks, where blocks are split into two groups based on an *admissibility condition*. Admissible blocks are stored as low-rank matrices, while inadmissible blocks are either further decomposed or stored as dense matrices. For more details on a construction of hierarchical matrices we refer to [67].

In addition to already mentioned discretizations of integral and partial different equations, a wide variety of applications include hierarchical matrices. In the following we provide a short list of applications; for an abundant overview we refer to [67]. In the context of Riccati and Lyapunov matrix equations, the computation of the sign function of an $\mathcal{H}$–matrix has been discussed in [9, 62], alongside with a divide-and-conquer method using hierarchical matrices [80]. The work in [55, 56, 62, 68] involves the $\mathcal{H}$–matrix approximation of resolvents, which is then used to compute the matrix exponential and related matrix functions. Methods for computing (partial) spectral decompositions in $\mathcal{H}$–matrix formats, including HODLR, HSS and $\mathcal{H}^2$–matrices, have been developed in [16, 18, 20, 70, 61, 111]. Hierarchical matrix techniques have been employed in preconditioning [54, 93] and for developing (sparse) direct solvers in [3, 57, 92, 97, 113, 115]. Several other applications include kernel approximation in machine learning [103, 112], as well as sparse covariance matrix estimation [2, 7] in the HODLR and $\mathcal{H}$–matrix formats.

Hierarchically semi-separable ($\mathcal{H}^2$–) matrices are a class of $\mathcal{H}$–matrices, where the general low-rank representation of the matrix sub-blocks is replaced by a nested multilevel representation that takes advantage of the relationships between different submatrices; see [29] and references therein for more details. This representation leads to reduced storage requirements and lower arithmetical cost of the matrix operations in comparison

to the $\mathcal{H}$–matrix format. However, the nestedness condition makes the implementation of algebraic operations significantly more involved than for other $\mathcal{H}$–matrix formats.

Hierarchically off-diagonal low-rank (HODLR) matrices form the simplest class of hierarchical matrices [67], with low-rank off-diagonal blocks at multiple levels. In the HODLR matrix structure only the diagonal blocks are further subdivided into low-rank and full-rank blocks, whereas for general $\mathcal{H}$–matrices the off-diagonal blocks are decomposed as well. Thus, the HODLR format yields a much simpler representation and is often used because of its simplicity compared to general $\mathcal{H}$–matrices.

Hierarchically semiseparable (HSS) matrices, proposed in [32, 33], are a class of HODLR matrices that allow a hierarchical representation based on a recursive row and column partitioning. Unlike for HODLR matrices, where the off-diagonal blocks are stored independently, HSS matrices have the off-diagonal blocks with hierarchically nested structure. This also implies that HSS matrices form a subset of $\mathcal{H}^2$–matrices.

In Figure 2.1 we show the relations between the mentioned hierarchical matrix structures. Although subsequent chapters *do not* involve computations with $\mathcal{H}^2$ and HSS matrices, because of their close relation with HODLR matrices, they are introduced for completeness.

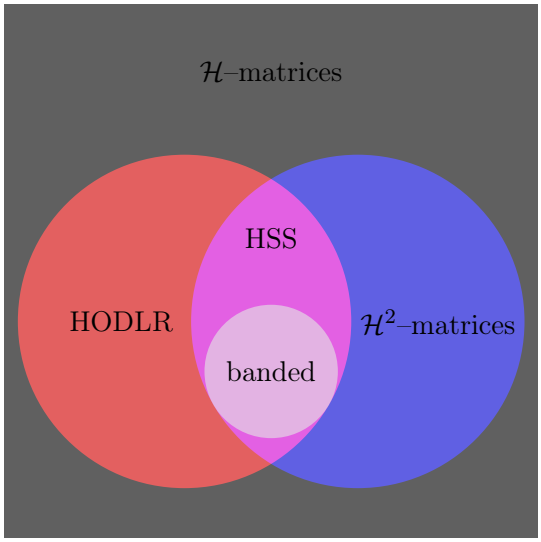

Figure 2.1 – An Euler diagram of different matrix structures, given that admissible blocks have ranks bounded by k.

2.1.1 HODLR matrices

Given an $n \times m$ matrix M let us consider a block matrix partitioning of the form

$$M = \left[\begin{array}{c|c} M_{11}^{(1)} & M_{12}^{(1)} \\ \hline M_{21}^{(1)} & M_{22}^{(1)} \end{array}\right], \tag{2.1}$$

where $M_{12}^{(1)}, M_{21}^{(1)}$ are matrices with low rank, and $M_{11}^{(1)}, M_{22}^{(1)}$ are dense matrices or matrices of the form (2.1). This partitioning is applied recursively p times to the diagonal blocks, until their sizes have reached a certain minimal block-size, leading to the hierarchical partitioning shown in Figure 2.2.

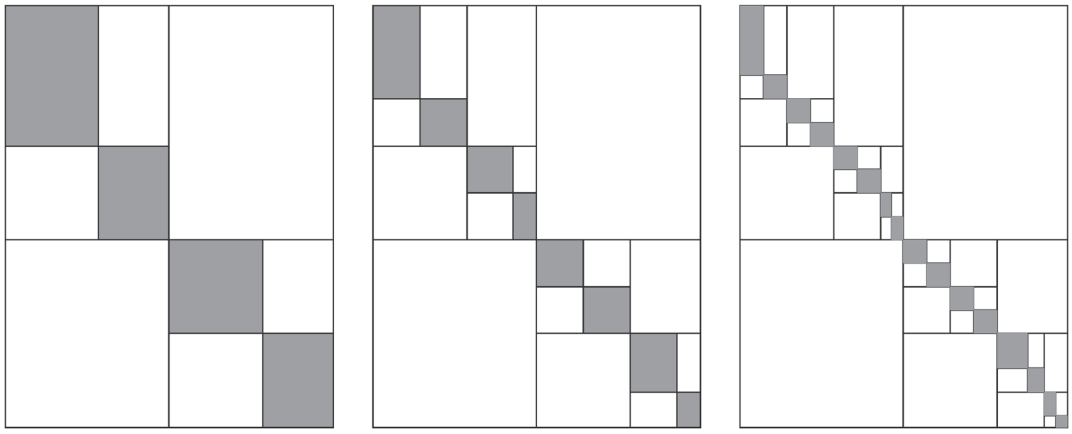

Figure 2.2 – Illustration of HODLR matrices for different recursion levels p. From left to right: $p = 2$, $p = 3$, and $p = 4$. The diagonal blocks (gray) are stored as dense matrices, while the off-diagonal blocks (white) are stored in terms of their low-rank factors.

We say that M is a *HODLR matrix* of level p and HODLR rank k if all off-diagonal blocks seen during this process have rank at most k. In the HODLR format, the off-diagonal blocks are stored, more efficiently, in terms of their low-rank factors. For example, for $p = 2$, the HODLR format takes the form

$$M = \left[\begin{array}{cc|c} M_{11}^{(2)} & U_1^{(2)}\left(V_2^{(2)}\right)^T & U_1^{(1)}\left(V_2^{(1)}\right)^T \\ U_2^{(2)}\left(V_1^{(2)}\right)^T & M_{22}^{(2)} & \\ \hline U_2^{(1)}\left(V_1^{(1)}\right)^T & & \begin{array}{cc} M_{33}^{(2)} & U_3^{(2)}\left(V_4^{(2)}\right)^T \\ U_4^{(2)}\left(V_3^{(2)}\right)^T & M_{44}^{(2)} \end{array} \end{array}\right].$$

The definition of a HODLR matrix depends on how the partitioning (2.1) is chosen on each level of the recursion. This choice is completely determined by the integer partitions

$$n = n_1 + n_2 + \cdots n_{2^p}, \qquad m = m_1 + m_2 + \cdots + m_{2^p}. \tag{2.2}$$

Depth p of the recursion and integers n_j, m_j, similar in size, are chosen such that n_j, m_j are nearly equal to a given minimal block-size $n_{\min}$, meaning that the integer partitions (2.2) are balanced. Here $n_j \times m_j$, $j = 1, \ldots, 2^p$, correspond to the sizes of the diagonal blocks $M_{11}^{(p)}, \ldots, M_{2^p,2^p}^{(p)}$ on the lowest level of the recursion, which are stored as dense matrices.

Given specific integer partitions (2.2), formally we define the set of HODLR matrices of level p and of HODLR rank k as

$$\mathcal{H}_{n\times m}(k) := \Big\{ M \in \mathbb{R}^{n\times m} : \operatorname{rank} M|_{\text{off}} \leq k \\ \forall \text{ off-diagonal block } M|_{\text{off}} \text{ in the recursive subdivision} \Big\}. \tag{2.3}$$

Remark 2.1. In practice, for a given matrix $M \in \mathbb{R}^{n\times m}$ and a prescribed minimal block-size $n_{\min}$, the integer partitions (2.2), and hence a specific HODLR format, are generated by imposing that matrices $M_{11}^{(1)}, M_{22}^{(1)}$ in (2.1) satisfy $M_{11}^{(1)} \in \mathbb{R}^{\lfloor \frac{n}{2} \rfloor \times \lfloor \frac{m}{2} \rfloor}$, $M_{22}^{(1)} \in \mathbb{R}^{\lceil \frac{n}{2} \rceil \times \lceil \frac{m}{2} \rceil}$, and by applying the same rule for sizes of the diagonal blocks in subsequent levels of the recursive subdivision. This choice of matrix division leads to a balanced partitioning of the matrix. In case $m = n = 2^p n_{\min}$, we get that the integer partitions (2.2) satisfy $m_i = n_i = n_{\min}$, for $i = 1, \ldots, 2^p$, yielding a perfectly balanced partitioning.

Computing HODLR representation

Given an $n \times m$ dense matrix M and integer partitions (2.2), the computation of its approximation in the HODLR format can be carried out in the following fashion. The diagonal blocks on the lowest level of recursion are stored explicitly, while low-rank representations of the off-diagonal blocks need to be computed.

A number of methods exist for computing a low-rank approximation of a matrix, such as adaptive cross approximation [67, Chapter 9], randomized techniques [74], rank-revealing algorithms [65], Krylov-based algorithms [104] and SVD-based algorithms [60, Chapter

2]; see also [8, 67] for more details and references.

In this work we compute low-rank representations using the truncated singular value decompositions of the off-diagonal blocks of M. For simplifying the presentation, we often assume that the ranks in the off-diagonal blocks are all bounded by the same integer k. For an arbitrary $A \in \mathbb{R}^{n\times m}$, let $A = U\Sigma V^T$ be an SVD decomposition of A, with $\Sigma = \operatorname{diag}(\sigma_1, \ldots, \sigma_{\min\{n,m\}})$, and $\sigma_1 \geq \sigma_2 \cdots \geq \sigma_{\min\{n,m\}}$. We define a truncation operator $\mathcal{T}_k$ as

$$\mathcal{T}_k(A) := U_k \Sigma_k V_k^T, \quad k \leq \min\{n, m\}, \tag{2.4}$$

where $U_k = (u_1, \ldots, u_k)$, $V_k = (v_1, \ldots, v_k)$ are left and right singular vectors of A, respectively, corresponding to the k largest singular values $\sigma_1, \ldots, \sigma_k$ and $\Sigma_k = \operatorname{diag}(\sigma_1, \ldots, \sigma_k)$. Then $\mathcal{T}_k(A)$ is the best rank-k approximation to A in the matrix 2-norm [79, Chapter 3]. Moreover, the truncated SVD $\mathcal{T}_k(A)$ yields a low-rank representation $\tilde{U}\tilde{V}^T$ of A given as

$$\tilde{U} := U_k \sqrt{\Sigma_k}, \quad \tilde{V} = V_k \sqrt{\Sigma_k},$$

under the assumption that $k \ll \min\{n, m\}$. Finally, this low-rank representation requires $(n + m)k$ units of storage, instead of nm when A is stored as a dense matrix.

In practice, we choose the ranks in the off-diagonal blocks adaptively based on an absolute truncation tolerance ϵ. To this end, we employ a truncation operator $\mathcal{T}_\epsilon$, defined as

$$\mathcal{T}_\epsilon(A) := \mathcal{T}_{k_\epsilon}(A), \qquad \text{with} \qquad k_\epsilon = \min_k \|\mathcal{T}_k(A) - A\|_2 \leq \epsilon. \tag{2.5}$$

Computing a low-rank representation from $\mathcal{T}_\epsilon(A)$ allows to choose the truncation rank k to attain a prescribed accuracy. However, using $\mathcal{T}_\epsilon$ possibly yields different ranks for each off-diagonal block in the HODLR format.

Storage requirements

Assume that the integer partitions (2.2) are balanced and $p = \mathcal{O}(\log n)$. For $M \in \mathcal{H}_{n\times m}(k)$ all off-diagonal blocks are stored in terms of their low-rank representations. In particular, from Figure 2.2 we can see that on level $0 < l \leq p$ the matrix M has 2^l off-diagonal blocks. Thus the storage complexity of the off-diagonal blocks on level l is $(n+m)k$. This

implies that the overall memory required for storing all off-diagonal blocks is $p(n+m)k$.

On the lowest level of recursion, the diagonal blocks are stored as $n_i \times m_i$ dense matrices, $i = 1, \ldots, 2^p$, and their storage requires

$$n_1 m_1 + \ldots n_{2^p} m_{2^p} = \mathcal{O}(n \cdot \max_i m_i)$$

units of memory. Thus, assuming that $\max_i m_i = \mathcal{O}(k)$, the storage complexity of a $n \times m$ HODLR matrix of rank k sums up to

$$\mathcal{O}(k\tilde{n} \log \tilde{n}), \quad \tilde{n} = \max\{n, m\}.$$

2.1.2 Hierarchical matrices

$\mathcal{H}$–matrices are a generalization of HODLR matrices that allow for more flexibility in the choice of blocks to be approximated by a low-rank matrix. In this section we focus on a particular variant of hierarchical matrices that is inspired by the discretization for 1D integral equations [66].

For simplicity, we assume $n = 2^p n_{\min}$ for some integers p and $n_{\min}$. Let $I = \{1, 2, \ldots, n\}$ denote the row and column index sets of a matrix $M \in \mathbb{R}^{n \times n}$. To consider more general hierarchical matrices, we define a partition P of $I \times I$ as follows. On level $l = 0$, the index set $I^0 := I$ is partitioned into $I^0 = I_1^1 \cup I_2^1$, with $I_1^1 = \{1, \ldots, \frac{n}{2}\}$ and $I_2^1 = \{\frac{n}{2} + 1, \ldots, n\}$. At this point, the partition P contains five blocks: $I \times I$ and $I_i^1 \times I_j^1$ for $i, j = 1, 2$. The subdivision continues as follows: on each level $l = 1, \ldots, p-1$ the index sets I_i^l are partitioned into sets I_{2i-1}^{l+1} and I_{2i}^{l+1} of equal size, contributing the blocks $I_i^{l+1} \times I_j^{l+1}$ for $i, j = 1, \ldots, 2^l$ to the partition P. The recursion terminates when a block $I_i^l \times I_j^l$ satisfies a certain admissibility condition or when $\min\{|I_i^l|, |I_j^l|\} \leq n_{\min}$ holds.

We consider the following variant of the so-called *standard admissibility condition*:

$$\text{block } \tau = t \times s \text{ is admissible} \iff \min\{\operatorname{diam}(t), \operatorname{diam}(s)\} \leq \operatorname{dist}(t, s), \tag{2.6}$$

where

$$\operatorname{diam}(t) := \max_{i,j \in t} |i - j|, \quad \operatorname{dist}(t, s) := \min_{i \in t, j \in s} |i - j|.$$

See Figure 2.3 for an illustration of the resulting partition P. Given P, the set of $\mathcal{H}$–matrices with block-wise rank k is defined as

$$\mathcal{H}(P,k) := \left\{ M \in \mathbb{R}^{n\times n} : \operatorname{rank} M|_\tau \leq k \text{ for all admissible blocks } \tau \in P \right\}.$$

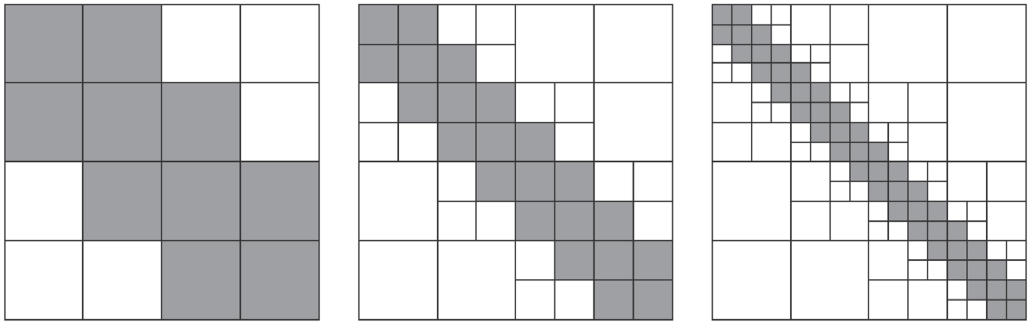

Figure 2.3 – Hierarchical matrices with admissibility condition (2.6) *for* $p = 2$, $p = 3$ *and* $p = 4$. *Blocks colored in grey are stored as dense matrices, while blocks colored in white are stored in terms of their low-rank factors.*

The resulting structure is very similar to the HODLR format, with a finer subdivision around the diagonal, allowing more matrices to be stored as dense.

Let us remark that the HODLR format likewise can be generated using an admissibility condition. Using the so-called *weak admissibility condition* [69], given as:

$$\text{block } \tau = t \times s \text{ is admissible } \iff t \neq s. \tag{2.7}$$

every off-diagonal block is considered admissible. This implies that the off-diagonal blocks are given in terms of their low-rank factors, which exactly corresponds to the HODLR format.

Example 2.2. (**Comparison between HODLR and $\mathcal{H}$–matrix format**) We investigate the potential of the HODLR and $\mathcal{H}$–matrix formats to efficiently store spectral projectors of banded matrices, whose computation is addressed in Chapter 4. For this purpose, we have generated, as explained in Section 4.5.1, a symmetric b-banded matrix $A \in \mathbb{R}^{16000\times 16000}$ with eigenvalues in $[-1,\ -\mathsf{gap}] \cup [\mathsf{gap},\ 1]$, such that half of the spectrum is negative. The memory needed to store the full spectral projector $\Pi_{<0}(A)$ associated with the negative eigenvalues in double precision is 2048 MB. We choose $n_{\min} = 250$, and obtain the integer partition generating the HODLR format as explained in Remark 2.1. Moreover, we use the truncation tolerance $\epsilon = 10^{-10}$, and $\mathsf{gap} \in \{10^{-1}, 10^{-4}\}$. Table 2.1

reveals that the HODLR format often requires less memory to store the approximation of $\Pi_{<0}(A)$, unless both gap and the bandwidth are large. However, in terms of computational time, the outcome is even clearer. For bandwidth $b = 8$ and $\mathsf{gap} = 10^{-1}$, a situation that favors the $\mathcal{H}$–matrix format in terms of memory in Table 2.1, we have run the algorithm described in Section 4.4 in both formats. It turned out that the use of the HODLR format led to an overall time of 608 seconds, while the $\mathcal{H}$–matrix format required 792 seconds.

Table 2.1 – Memory required to approximately store spectral projectors for the banded matrices in HODLR and $\mathcal{H}$–matrix format.

$\mathsf{gap} = 10^{-1}$	HODLR	$\mathcal{H}$–matrix
b = 1	55.72 MB	95.16 MB
b = 2	79.38 MB	96.42 MB
b = 4	127.04 MB	106.54 MB
b = 8	219.92 MB	151.06 MB
b = 16	395.91 MB	291.85 MB

$\mathsf{gap} = 10^{-4}$	HODLR	$\mathcal{H}$–matrix
b = 1	86.03 MB	128.58 MB
b = 2	129.71 MB	160.56 MB
b = 4	206.32 MB	225.72 MB
b = 8	340.88 MB	352.54 MB
b = 16	567.69 MB	583.93 MB

Based on the evidence provided by Example 2.2, we have concluded that more general $\mathcal{H}$–matrix formats bring little advantage and therefore focus on the HODLR format in the subsequent chapters.

2.1.3 Hierarchically semiseparable matrices

As aforementioned, hierarchically semiseparable matrices are a special case of HODLR matrices. Starting from a given HODLR format generated by the integer partition (2.2), we consider an off-diagonal block of a HODLR matrix M on level $0 < l < p$. Assuming that the off-diagonal blocks have exact rank k, their low-rank representations can be also written as

$$M_{ij}^{(l)} = U_i^{(l)} S_{ij}^{(l)} \left(V_j^{(l)}\right)^T, \quad \text{with } S_{ij}^{(l)} \in \mathbb{R}^{k\times k}, \quad \text{for } i \neq j.$$

HSS matrices additionally satisfy that the ases for the off-diagonal blocks on different levels are nested. More specifically, there exist matrices $R_i^{(l)}, W_j^{(l)} \in \mathbb{R}^{2k\times k}$ such that

$$U_i^{(l)} = \begin{bmatrix} U_{2i-1}^{(l+1)} & 0 \\ 0 & U_{2i}^{(l+1)} \end{bmatrix} R_i^{(l)}, \text{ with } R_i^{(l)} = \begin{bmatrix} R_{i,2i-1}^{(l)} \\ R_{i,2i}^{(l)} \end{bmatrix}, \tag{2.8}$$

$$V_j^{(l)} = \begin{bmatrix} V_{2j-1}^{(l+1)} & 0 \\ 0 & V_{2j}^{(l+1)} \end{bmatrix} W_j^{(l)}, \text{ with } W_j^{(l)} = \begin{bmatrix} W_{j,2j-1}^{(l)} \\ W_{j,2j}^{(l)} \end{bmatrix}. \tag{2.9}$$

Using the conditions (2.8) and (2.9), the row and column bases $U^{(l)}$ and $V^{(l)}$ can be constructed from the bases on the highest level, i.e., the bases on levels $0 < l < p$ can be retrieved recursively from the bases $U_i^{(p)}$ and $V_j^{(p)}$ and the matrices $R_i^{(l)}$ and $W_j^{(l)}$. For instance, for $p = 2$, the HSS format takes the form

$$M = \left[\begin{array}{cc|c} \begin{bmatrix} M_{11}^{(2)} & U_1^{(2)} S_{12}^{(2)} \left(V_2^{(2)}\right)^T \\ U_2^{(2)} S_{21}^{(2)} \left(V_1^{(2)}\right)^T & M_{22}^{(2)} \end{bmatrix} & & \begin{bmatrix} U_1^{(2)} R_{11}^{(1)} \\ U_2^{(2)} R_{12}^{(1)} \end{bmatrix} S_{12}^{(1)} \begin{bmatrix} V_3^{(2)} W_{21}^{(1)} \end{bmatrix}^T \\ \hline & \begin{bmatrix} M_{33}^{(2)} & U_3^{(2)} S_{34}^{(2)} \left(V_4^{(2)}\right)^T \\ U_4^{(2)} S_{43}^{(2)} \left(V_3^{(2)}\right)^T & M_{44}^{(2)} \end{bmatrix} \end{array}\right].$$

This means that the storage requirements of an HSS matrix is determined by the cost of storing the dense diagonal blocks $M_{ii}^{(p)}$ on the highest level, the bases $U_i^{(p)}$ and $V_j^{(p)}$, likewise $R_i^{(l)}, W_j^{(l)}$ and $S_{2i-1,2i}^{(l)}, S_{2i,2i-1}^{(l)}$. Thus the storage complexity for a matrix in the HSS format with rank k sums up to $\mathcal{O}(kn)$. Compared to the storage of a HODLR matrix of rank k, a $\log n$ factor can be avoided as a result of the nestedness condition.

For a b-banded matrix M its HSS representation with rank k can be computed exactly, with $k = 2b$. The following result relates representations in the HODLR and HSS formats for general matrices.

Lemma 2.3. (Lemma 2.5.9 in [85]) *Let $M \in \mathbb{R}^{n \times n}$, where n satisfies the integer partition* (2.2)*. Then:*

i) If M is an HSS matrix of level p with rank k, then M is a HODLR matrix of level p with rank k.

ii) If M a HODLR matrix of level p with rank k, then M is an HSS of level p with rank kp.

Similarly as for other hierarchical formats, the HSS format allows for approximate arithmetic operations. We refer the reader to [33, 115] for a detailed description of hierarchically semiseparable matrices and the associated arithmetics. Compared to the HSS format, the HODLR matrices are much simpler as they lack the nested off-diagonal bases. However, this makes the implementation of the arithmetic operations in the HODLR format less tedious and more straightforward.

Example 2.4. (**Comparison between HODLR and HSS formats**) We also provide a comparison of the storage requirements in the HODLR and the HSS formats. We first consider spectral projectors of banded matrices of size $n = 8000$, generated as explained in Section 4.5.1. The eigenvalues of the considered matrices are in $[-1, -10^{-1}] \cup [10^{-1}, 1]$. We compute both the HODLR and the HSS representation of the spectral projectors using the truncation tolerance $\epsilon = 10^{-10}$. In the second case, we investigate the storage of a random matrix of size $n = 8000$ with a prescribed off-diagonal rank k. This means that these matrices can be stored exactly in the HODLR and the HSS formats, therefore compression is not performed. In both cases, the minimal block-size is set to $n_{\min} = 250$, and the obtained integer partition is as in Remark 2.1. To compute an HSS representation of a matrix, we make use of the `hm-toolbox`, available at https://github.com/numpi/hm-toolbox. In Table 2.2 (left) we see the advantages of the HSS format in terms of storage for banded matrices for smaller bandwidths. Additionally, Table 2.2 (right) shows that for the considered n, as the prescribed rank increases, the benefits of the HSS format become insignificant compared to the HODLR format.

Table 2.2 – Left: Memory required to store spectral projectors of banded matrices in the HODLR and the HSS formats. Right: Memory required to store random matrices with the off-diagonal rank k in the HODLR and the HSS formats.

band	HODLR	HSS
b = 1	26.8 MB	21.4 MB
b = 2	36.9 MB	29.1 MB
b = 4	53.0 MB	47.8 MB
b = 8	89.4 MB	103.8 MB
b = 16	159.0 MB	213.0 MB

random	HODLR	HSS
k = 1	15.9 MB	15.9 MB
k = 10	21.4 MB	23.6 MB
k = 20	27.5 MB	36.4 MB
k = 40	39.7 MB	75.3 MB
k = 80	64.1 MB	141.2 MB

2.2 HODLR arithmetics

Several arithmetic operations can be performed efficiently in the HODLR format. However, the need for truncation in addition, multiplication, inversion and computation of Cholesky or LU decomposition in order to keep the ranks bounded, makes the arithmetic involving HODLR matrices an approximative one. Because of their structure, operations with HODLR matrices are performed recursively in a block-wise manner, resulting in a linear-polylogarithmic complexity.

In order to simplify the presentation we use the following assumptions. Firstly, we assume

that for the size of HODLR matrices it holds $n = 2^p n_{\min}$. Moreover, we assume the integer partitions explained in Remark 2.1. The complexity analysis is derived under the assumption that the off-diagonal ranks are bounded by a constant k.

2.2.1 Preliminaries on low-rank matrix arithmetics

As low-rank matrices are a fundamental tool when dealing with arithmetic operations in the HODLR format, we first recall several operations with low-rank matrices. We assume that low-rank matrices are given in terms of their low-rank factors, and that their rank is significantly smaller than the matrix size.

Matrix-vector multiplication. Let $A = U_A V_A^T$, with $U_A \in \mathbb{R}^{n \times k_A}, V_A \in \mathbb{R}^{m \times k_A}$, and $x \in \mathbb{R}^m$. The multiplication $y = Ax$ is performed in the following two steps:

1. $\tilde{y} = V_A^T x$,
2. $y = U_A \tilde{y}$.

This leads to a computational cost $2(n + m)k_A$.

Matrix-matrix multiplication. Let $A = U_A V_A^T, B = U_B V_B^T$ with $U_A \in \mathbb{R}^{n \times k_A}, V_A \in \mathbb{R}^{m \times k_A}$ and $U_B \in \mathbb{R}^{m \times k_B}, V_B \in \mathbb{R}^{l \times k_B}$. We note that there are two possible low-rank representations of the product AB:

1. $AB = (U_A V_A^T U_B) V_B^T$, calculated with $2(m + l)k_A k_B$ operations,
2. $AB = U_A (V_A^T U_B V_B^T)$, calculated with $2(n + m)k_A k_B$ operations.

Matrix compression. Given $A = U_A V_A^T$, with $U_A \in \mathbb{R}^{n \times k_A}, V_A \in \mathbb{R}^{m \times k_A}$ and $k < k_A \ll \min\{n, m\}$, the aim is to compute a rank-k approximation to A. This can be done by using the following procedure.

1. Compute $U_A = Q_U R_U$ and $V_A = Q_U R_U$, the economic QR decompositions of U_A and V_A, respectively.

2. Compute the truncated SVD of a $k_A \times k_A$ matrix: $\mathcal{T}_k(R_U R_V^T) = \tilde{U}\Sigma\tilde{V}$, with $\mathcal{T}_k$ defined in (2.4).

3. Set $U_A = Q_U\tilde{U}\sqrt{\Sigma}$ and $V_A = Q_V\tilde{V}\sqrt{\Sigma}$.

The described procedure requires $\mathcal{O}\left((n+m)k_A^2\right)$ operations to perform two QR decompositions, and additionally $\mathcal{O}(k_A^3)$ operations for the computation of the truncated SVD. Hence, the overall cost is linear in n and m.

The truncation with respect to a prescribed tolerance ϵ is performed in an analogous way, by employing the operator $\mathcal{T}_\epsilon$ given in (2.5) in place of $\mathcal{T}_k$.

Matrix addition. For $A = U_A V_A^T$ and $B = U_B V_B^T$ with $U_A \in \mathbb{R}^{n\times k_A}, V_A \in \mathbb{R}^{m\times k_A}$ and $U_B \in \mathbb{R}^{n\times k_B}, V_B \in \mathbb{R}^{m\times k_B}$, the sum of A and B can again be represented as a low-rank matrix:

$$A + B = [U_A\, U_B][V_A\, V_B]^T. \tag{2.10}$$

In this case, no arithmetic operations need to be performed. However, the rank of the resulting representation increases to $k_A + k_B$. If $k_A + k_B$ is larger than the exact rank of $A + B$, or larger than a prescribed rank k, it is beneficial to perform a compression of the low-rank representation in (2.10). Thus when truncation is involved, we say that a *formatted addition* is performed. In this case, the cost of adding two low-rank matrices corresponds to the cost of compression, i.e., $\mathcal{O}\left((k_A + k_B)^3 + (n+m)(k_A + k_B)^2\right)$.

2.2.2 HODLR matrix-vector multiplication

Given a HODLR matrix $M \in \mathcal{H}_{n\times n}(k)$ and a vector $v \in \mathbb{R}^n$, a matrix-vector multiplication $y = Mv$ can be performed without involving the truncation operation, i.e., the result is exact up to machine precision. For a level p HODLR matrix

$$M = \begin{bmatrix} M_{11} & U_1 V_2^T \\ U_2 V_1^T & M_{22} \end{bmatrix}$$

and a vector $v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$ partitioned accordingly, the resulting vector $y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$ can be computed as

$$y_1 = M_{11}v_1 + U_1(V_2^T v_2), \tag{2.11}$$
$$y_2 = M_{22}v_2 + U_2(V_1^T v_1). \tag{2.12}$$

We have two types of operations in (2.11) and (2.12):

1. the multiplication of a level $p-1$ HODLR matrix with a vector, and

2. the multiplication of a low-rank matrix with a vector.

In case 1, we proceed recursively until level 0 HODLR matrices have been reached. Thus, at the end of the recursion a standard matrix-vector multiplication is performed, whereas in case 2 low-rank arithmetics is used.

Let $\mathcal{C}_{Hv}(n)$ denote the computational cost of a HODLR matrix-vector multiplication of size n. Using the results from Section 2.2.1, it follows that the overall computational cost of a matrix-vector multiplication satisfies the recursion

$$\mathcal{C}_{Hv}(n) = \begin{cases} 2n_{\min}, & \text{if } n = n_{\min}\ (p = 0), \\ 2\mathcal{C}_{Hv}\left(\frac{n}{2}\right) + (4k+1)n, & \text{otherwise.} \end{cases}$$

Then from the Master theorem [34, Chapter 4], we conclude that the computational complexity of a matrix-vector multiplication is

$$\mathcal{C}_{Hv}(n) \in \mathcal{O}(kn \log n). \tag{2.13}$$

The pseudocode is presented in Algorithm 2.1. For completeness, we note that the multiplication with a (low-rank) matrix $V \in \mathbb{R}^{n \times l}$ is performed analogously, and that its cost $\mathcal{C}_{HR}(n)$ satisfies $\mathcal{C}_{HR}(n) = l \cdot \mathcal{C}_{Hv}(n)$. Moreover, similar recursive procedures can be derived for the multiplication from the left with a vector and with a (low-rank) matrix.

Algorithm 2.1 Matrix-vector multiplication

Input: $M \in \mathcal{H}_{n\times n}(k)$ of level p, $v \in \mathbb{R}^n$.
Output: Vector $y = Mv \in \mathbb{R}^n$.

1: **function** $y = \texttt{hMV}(M, v)$
2: **if** $p = 0$ **then**
3: Return $y = Mv$.
4: **else**
5: Partition $M = \begin{bmatrix} M_{11} & U_1V_2^T \\ U_2V_1^T & M_{22} \end{bmatrix}$ and $v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$ conformally.
6: Call recursively $y_1 = \texttt{hMV}(M_{11}, v_1) + U_1(V_2^T v_2)$.
7: Call recursively $y_2 = \texttt{hMV}(M_{22}, v_2) + U_2(V_1^T v_1)$.
8: Return $y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$.
9: **end if**
10: **end function**

2.2.3 HODLR matrix addition

Given two HODLR matrices $M, N \in \mathcal{H}_{n\times n}(k)$ of level p, we consider the computation of $A = M + N$. The sum of HODLR matrices can be represented in the HODLR format as

$$M+N = \begin{bmatrix} M_{11} & U_1V_2^T \\ U_2V_1^T & M_{22} \end{bmatrix} + \begin{bmatrix} N_{11} & X_1Y_2^T \\ X_2Y_1^T & N_{22} \end{bmatrix} = \begin{bmatrix} M_{11}+N_{11} & U_1V_2^T + X_1Y_2^T \\ U_2V_1^T + X_2Y_1^T & M_{22}+N_{22} \end{bmatrix},$$

where $M_{11}, M_{22}, N_{11}, N_{22}$ are HODLR matrices of size $n/2$ and level $p-1$. The diagonal blocks are computed recursively, whilst for the off-diagonal blocks the low-rank arithmetics is used. In particular, two types of operations are employed when performing matrix addition:

1. the addition of two HODLR matrices of size $n/2$,
2. the addition of two rank-k matrices of size $n/2$.

In case 1 recursive calls continue until level 0 HODLR matrices have been reached. On the lowest level of the recursion, a standard matrix addition is used, with the cost $n_{\min}^2$. In case 2 we perform the formatted addition of two low-rank matrices, explained in Section 2.2.1. This guarantees that all off-diagonal ranks in the sum are bounded by k. Finally, this leads to the pseudocode given in Algorithm 2.2.

Algorithm 2.2 Matrix addition ($+_{\mathcal{H}}$)

Input: $M, N \in \mathcal{H}_{n\times n}(k)$ of level p.
Output: $A = M +_{\mathcal{H}} N \in \mathcal{H}_{n\times n}(k)$ of level p.

1: **if** $p = 0$ **then**
2: Return $A = M + N$.
3: **else**
4: Partition $M = \begin{bmatrix} M_{11} & U_1V_2^T \\ U_2V_1^T & M_{22} \end{bmatrix}$ and $N = \begin{bmatrix} N_{11} & X_1Y_2^T \\ X_2Y_1^T & N_{22} \end{bmatrix}$.
5: Call recursively $A_{11} = M_{11} +_{\mathcal{H}} N_{11}$.
6: Call recursively $A_{22} = M_{22} +_{\mathcal{H}} N_{22}$.
7: Compute $Z_1W_2^T = \mathcal{T}_k([U_1\, X_1][V_2\, Y_2]^T)$.
8: Compute $Z_2W_1^T = \mathcal{T}_k([U_2\, X_2][V_1\, Y_1]^T)$.
9: Return $A = \begin{bmatrix} A_{11} & Z_1W_2^T \\ Z_2W_1^T & A_{22} \end{bmatrix}$.
10: **end if**

The complexity of a HODLR matrix addition of size n, denoted by $\mathcal{C}_{H+H}(n)$, thus satisfies the recurrence

$$\mathcal{C}_{H+H}(n) = \begin{cases} n_{\min}^2, & \text{if } n = n_{\min}\ (p = 0), \\ 2\mathcal{C}_{H+H}\left(\frac{n}{2}\right) + \mathcal{O}(k^3 + k^2 n), & \text{otherwise.} \end{cases}$$

Lastly, using the Master theorem, we get

$$\mathcal{C}_{H+H}(n) \in \mathcal{O}(k^3 n + k^2 n \log n). \tag{2.14}$$

Similarly, the sum of a HODLR matrix and a low-rank matrix can be computed in the HODLR format. In order to perform the addition, we first need to partition the low-rank matrix according to the HODLR matrix. Let $M \in \mathcal{H}_{n\times n}(k)$ and $R = AB^T$, with $A, B \in \mathbb{R}^{n\times k}$. Then

$$M+R = \begin{bmatrix} M_{11} & U_1V_2^T \\ U_2V_1^T & M_{22} \end{bmatrix} + \begin{bmatrix} (AB^T)_{11} & (AB^T)_{12} \\ (AB^T)_{21} & (AB^T)_{22} \end{bmatrix} = \begin{bmatrix} M_{11} + (AB^T)_{11} & U_1V_2^T + (AB^T)_{12} \\ U_2V_1^T + (AB^T)_{21} & M_{22} + (AB)_{22}^T \end{bmatrix}.$$

The computation of the diagonal and off-diagonal blocks in the sum is processed as

aforementioned. The computational cost $\mathcal{C}_{H+R}(n)$ of the addition satisfies

$$\mathcal{C}_{H+R}(n) = \begin{cases} (2k+1)n_{\min}^2, & \text{if } n = n_{\min}\ (p=0), \\ 2\mathcal{C}_{H+R}\left(\frac{n}{2}\right) + \mathcal{O}(k^3 + k^2 n), & \text{otherwise.} \end{cases}$$

Thus the asymptotic complexity coincides with the complexity $\mathcal{C}_{H+H}(n)$.

2.2.4 HODLR matrix-matrix multiplication

We now have all the needed ingredients to compute a product of two HODLR matrices. Suppose $M, N \in \mathcal{H}_{n\times n}(k)$ of level p. Then, the computation of their product is carried out block-wise:

$$\begin{aligned} MN &= \begin{bmatrix} M_{11} & U_1 V_2^T \\ U_2 V_1^T & M_{22} \end{bmatrix} \begin{bmatrix} N_{11} & X_1 Y_2^T \\ X_2 Y_1^T & N_{22} \end{bmatrix} \\ &= \begin{bmatrix} M_{11}N_{11} + U_1 V_2^T X_2 Y_1^T & M_{11} X_1 Y_2^T + U_1 V_2^T N_{22} \\ U_2 V_1^T N_{11} + M_{22} X_2 Y_1^T & U_2 V_1^T X_1 Y_2^T + M_{22} N_{22} \end{bmatrix}, \end{aligned}$$

where $M_{11}, M_{22}, N_{11}, N_{22}$ are HODLR matrices of size $n/2$ and level $p-1$. The ansatz is that the resulting matrix has the same block structure as M and N. The following operations are involved when computing the product of two HODLR matrices:

1. the multiplication of two HODLR matrices,
2. the multiplication of a HODLR matrix with a low-rank matrix,
3. the multiplication of two low-rank matrices,
4. the addition of a HODLR matrix and a low-rank matrix,
5. the addition of two low-rank matrices.

The operations in 2, 3, 4 and 5 have already been discussed, together with their computational cost. As stated in Section 2.2.2 and Section 2.2.1, the computations in 2 and 3, respectively, are performed exactly, since those operations do not increase ranks. On the other hand, as showed in the previous section, performing the computations in 4 and 5 involves the truncation operator $\mathcal{T}_k$, in order to keep the ranks bounded. The computation

in 1 is carried out recursively, and performs a standard matrix-matrix multiplication on level 0 HODLR matrices, with the computational cost $\mathcal{O}(n_{\min}^3)$. A pseudocode of the procedure is given in Algorithm 2.3.

Algorithm 2.3 Matrix multiplication ($*_{\mathcal{H}}$)

Input: HODLR matrices $M, N \in \mathcal{H}_{n\times n}(k)$ of level p.
Output: $A = M *_{\mathcal{H}} N \in \mathcal{H}_{n\times n}(k)$.

1: **if** $p = 0$ **then**
2: Return $A = MN$.
3: **else**
4: Partition $M = \begin{bmatrix} M_{11} & U_1V_2^T \\ U_2V_1^T & M_{22} \end{bmatrix}$ and $N = \begin{bmatrix} N_{11} & X_1Y_2^T \\ X_2Y_1^T & N_{22} \end{bmatrix}$.
5: Call recursively $A_{11} = M_{11} *_{\mathcal{H}} N_{11}$.
6: Compute $A_{11} \leftarrow A_{11} +_{\mathcal{H}} U_1(V_2^T X_2)Y_1^T$.
7: Compute $Z_1W_2^T = \mathcal{T}_k([\texttt{hMV}(M_{11}, X_1)\, U_1][Y_2\, \texttt{hMV}(N_{22}^T, V_2)])$.
8: Compute $Z_2W_1^T = \mathcal{T}_k([\texttt{hMV}(M_{22}, X_2)\, U_2][Y_1\, \texttt{hMV}(N_{11}^T, V_1)])$.
9: Call recursively $A_{22} = M_{22} *_{\mathcal{H}} N_{22}$
10: Compute $A_{22} \leftarrow A_{22} +_{\mathcal{H}} U_2(V_1^T X_1)Y_2^T$.
11: Return $A = \begin{bmatrix} A_{11} & Z_1W_2^T \\ Z_2W_1^T & A_{22} \end{bmatrix}$.
12: **end if**

Let $\mathcal{C}_{HH}(n)$ designate the computational cost of the HODLR matrix-matrix multiplication of size n. The previous developments lead to

$$\mathcal{C}_{HH}(n) = \begin{cases} \mathcal{O}(n_{\min}^3), & \text{if } n = n_{\min}\ (p = 0), \\ 2\mathcal{C}_{HH}\left(\frac{n}{2}\right) + \mathcal{O}(k^3 n + k^2 n \log n) + \mathcal{O}(k^3 + k^2 n), & \text{otherwise.} \end{cases}$$

Finally, we derive that

$$\mathcal{C}_{HH}(n) \in \mathcal{O}(k^3 n \log n + k^2 n \log^2 n). \tag{2.15}$$

In addition to the presented matrix-matrix multiplication, we will also consider the multiplication of rectangular HODLR matrices. Assuming that the partitions of the involved matrices are compatible, the multiplication in the HODLR format is completed analogously as for square matrices. Moreover, the asymptotic computational cost amounts to $\mathcal{C}_{HH}(n)$.

2.2.5 HODLR triangular linear systems

Solving triangular linear systems is one of the operations that can be performed efficiently in the HODLR format. In the following we focus on solving lower triangular linear systems. Analogous developments and analysis follow for upper triangular systems as well.

Linear systems with dense right-hand side

Given a lower triangular invertible matrix $M \in \mathcal{H}_{n\times n}(k)$ and a vector $y \in \mathbb{R}^n$, the solution of a system $Mx = y$ can be computed exactly in the HODLR format. We derive a procedure for solving a linear system by considering partitions of M, x and y

$$M = \begin{bmatrix} M_{11} & 0 \\ U_2 V_1^T & M_{22} \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix},$$

with M_{11}, M_{22} HODLR matrices of size $n/2$ and level $p-1$. Thus the procedure, summarized in Algorithm 2.4, consists of the following steps:

- solve recursively the lower triangular system

$$M_{11}x_1 = y_1 \tag{2.16}$$

- solve recursively the lower triangular system

$$M_{22}x_2 = y_2 - U_2(V_1^T x_1). \tag{2.17}$$

The computation of the right-hand side in (2.17) is executed exactly, as only the multiplication of a low-rank matrix with a vector is needed, alongside with the vector subtraction. As seen in Section 2.2.1, these operations can be performed with $(2k+1)n$ operations. The recursive calls in (2.16) and (2.17) are done using forward substitution, until the considered HODLR matrices are of level $p = 0$. In the latter case, a standard dense solver for forward substitution of complexity $\mathcal{O}(n_{\text{min}}^2)$ is applied. Hence, solving a triangular HODLR system with a dense right-hand side does not require rank truncation.

Let $\mathcal{C}_{\text{TSolve}}(n)$ denote the computational cost of the summarized procedure. Then

$$\mathcal{C}_{\text{TSolve}}(n) = \begin{cases} \mathcal{O}(n_{\min}^2), & \text{if } n = n_{\min}\ (p = 0), \\ 2\mathcal{C}_{\text{TSolve}}\left(\frac{n}{2}\right) + (2k+1)n, & \text{otherwise.} \end{cases}$$

Finally it follows

$$\mathcal{C}_{\text{TSolve}}(n) \in \mathcal{O}(kn \log n). \tag{2.18}$$

Algorithm 2.4 Solving a triangular system with a dense right-hand side

Input: Lower triangular invertible $M \in \mathcal{H}_{n\times n}(k)$ of level p, and $y \in \mathbb{R}^n$.
Output: Solution $x \in \mathbb{R}^n$ of a linear system $Mx = y$.

1: **function** $x = \texttt{TSolve}(M, y)$
2: **if** $p = 0$ **then**
3: Return $x = M \backslash y$.
4: **else**
5: Partition $M = \begin{bmatrix} M_{11} & 0 \\ U_2 V_1^T & M_{22} \end{bmatrix}$ and $y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$ conformally.
6: Call recursively $y_1 = \texttt{TSolve}(M_{11}, y_1)$.
7: Compute $S = y_2 - U_2(V_1^T x_1)$.
8: Call recursively $x_2 = \texttt{TSolve}(M_{22}, S)$.
9: Return $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$.
10: **end if**
11: **end function**

When the right-hand side is a matrix $Y \in \mathbb{R}^{n\times l}$, an analogous procedure and complexity estimates can be derived. This leads to an algorithm with the computational complexity

$$\mathcal{C}_{\text{TSolve},l}(n) \in \mathcal{O}(kln \log n). \tag{2.19}$$

Linear systems with HODLR right-hand side

Sometimes it is of interest to solve a triangular linear system when the right-hand side is a HODLR matrix. For a given invertible lower triangular $M \in \mathcal{H}_{n\times n}(k)$, and $Y \in \mathcal{H}_{n\times n}(k)$, suppose we aim to find a solution $X \in \mathcal{H}_{n\times n}(k)$ of a system $MX = Y$. Block-wise

partitions of M, X and Y

$$M = \begin{bmatrix} M_{11} & 0 \\ U_2 V_1^T & M_{22} \end{bmatrix}, \quad X = \begin{bmatrix} X_{11} & X_1 Y_2^T \\ X_2 Y_1^T & X_{22} \end{bmatrix}, \quad Y = \begin{bmatrix} Y_{11} & Z_1 W_2^T \\ Z_2 W_1^T & Y_{22} \end{bmatrix},$$

yield four matrix equations. Thus the solution X can be computed block-wise as:

- solve recursively the lower triangular system for the diagonal block X_{11}

$$M_{11} X_{11} = Y_{11}, \tag{2.20}$$

- solve the lower triangular systems for the off-diagonal blocks

$$M_{11} X_1 Y_2^T = Z_1 W_2^T, \tag{2.21}$$
$$M_{22} X_2 Y_1^T = \mathcal{T}_k([Z_2\, U_2][W_1\ -X_{11}^T V_1]). \tag{2.22}$$

- solve recursively the lower triangular system for the diagonal block X_{22}

$$M_{22} X_{22} = Y_{22} -_{\mathcal{H}} U_2 V_1^T X_1 Y_2^T. \tag{2.23}$$

The computation of the off-diagonal blocks in (2.21) and (2.22) can be carried out with Algorithm 2.4. Here, instead of a vector as a right-hand side we take a (low-rank) matrix. Thus the complexity of (2.21) and (2.22) amounts to $2\mathcal{C}_{\text{TSolve},k}\,(n/2) \in \mathcal{O}(k^2 n \log n)$, where we also count the compression of the right-hand side in (2.22) that requires $\mathcal{O}(k^3 + k^2 n)$ arithmetic operations. The recursive calls in (2.20) and (2.23) are continued until level 0 HODLR matrices have been reached, where the complexity of performing a standard forward substitution is $\mathcal{O}(n_{\min}^3)$. Algorithm 2.5 summarizes the described procedure.

Let $\mathcal{C}_{\text{hTSolve}}(n)$ denote the computational cost of Algorithm 2.5. Then the following recursion holds

$$\mathcal{C}_{\text{hTSolve}}(n) = \begin{cases} \mathcal{O}(n_{\min}^3), & \text{if } n = n_{\min}\ (p = 0), \\ 2\mathcal{C}_{\text{hTSolve}}\left(\frac{n}{2}\right) + \mathcal{O}(k^3 n + k^2 n \log n), & \text{otherwise.} \end{cases}$$

Algorithm 2.5 Solving a triangular system with a HODLR right-hand side

Input: Lower triangular invertible $M \in \mathcal{H}_{n\times n}(k)$ of level p, and $Y \in \mathcal{H}_{n\times n}(k)$.
Output: Solution $X \in \mathcal{H}_{n\times n}(k)$ of a linear system $MX = Y$.

1: **function** $X = \texttt{hTSolve}(M, Y)$
2: **if** $p = 0$ **then**
3: Return $X = M \backslash Y$.
4: **else**
5: Partition $M = \begin{bmatrix} M_{11} & 0 \\ U_2 V_1^T & M_{22} \end{bmatrix}$ and $Y = \begin{bmatrix} Y_{11} & Z_1 W_2^T \\ Z_2 W_1^T & Y_{22} \end{bmatrix}$ conformally.
6: Call recursively $X_{11} = \texttt{hTSolve}(M_{11}, Y_{11})$.
7: Compute $X_1 = \texttt{TSolve}(M_{11}, Z_1)$ and set $Y_2 = W_2$.
8: Compute $\tilde{Z}_2 \tilde{W}_1^T = \mathcal{T}_k([Z_2\, U_2][W_1\, -X_{11}^T V_1])$.
9: Compute $X_2 = \texttt{TSolve}(M_{22}, \tilde{Z}_2)$ and set $Y_1 = \tilde{W}_1$.
10: Compute $S = Y_{22} -_{\mathcal{H}} U_2 V_1^T X_1 Y_2^T$.
11: Call recursively $X_{22} = \texttt{hTSolve}(M_{22}, S)$.
12: Return $X = \begin{bmatrix} X_{11} & X_1 Y_2^T \\ X_2 Y_1^T & X_{22} \end{bmatrix}$.
13: **end if**
14: **end function**

Finally, from the Master theorem we retrieve

$$\mathcal{C}_{\text{hTSolve}}(n) \in \mathcal{O}(k^3 n \log n + k^2 n \log^2 n). \tag{2.24}$$

2.2.6 HODLR Cholesky decomposition

The hierarchical Cholesky decomposition is an approximate triangular factorization. As in the standard case, this factorization decomposes a HODLR matrix into a product of a lower and an upper triangular matrix.

In the following, we discuss an inexact computation of the Cholesky decomposition for a symmetric positive definite HODLR matrix $M \in \mathcal{H}_{n\times n}(k)$ of level p. The Cholesky factor is also represented in the HODLR format, as an upper triangular matrix, with the same hierarchical structure as M. Starting from the partition of M and the decomposition

$$M = \begin{bmatrix} M_{11} & U_1 V_2^T \\ V_2 U_1^T & M_{22} \end{bmatrix} = R^T R = \begin{bmatrix} R_{11}^T & 0 \\ Y_2 X_1^T & R_{22}^T \end{bmatrix} \begin{bmatrix} R_{11} & X_1 Y_2^T \\ 0 & R_{22} \end{bmatrix},$$

we get three equalities:

$$M_{11} = R_{11}^T R_{11}, \quad U_1 V_2^T = R_{11}^T X_1 Y_2^T, \quad M_{22} = R_{22}^T R_{22} + Y_2 X_1^T X_1 Y_2^T,$$

where M_{11} and M_{22} are HODLR matrices of size $n/2$ and level $p-1$. If the computations in (2.25) and (2.27) can be carried out, the computation of the approximate Cholesky decomposition is performed as follows:

- compute recursively the Cholesky factor R_{11} of M_{11}

$$M_{11} = R_{11}^T R_{11}, \tag{2.25}$$

- set $Y_2 = V_2$ and solve for X_1 by forward substitution

$$R_{11}^T X_1 = U_1, \tag{2.26}$$

- compute the Cholesky factor R_{22} of the Schur complement

$$R_{22}^T R_{22} = M_{22} -_{\mathcal{H}} Y_2 X_1^T X_1 Y_2^T. \tag{2.27}$$

Again, we note that the solution in (2.26) is exact, as no truncation operator is involved. When computing the Schur complement in (2.27), we perform a truncated subtraction to keep ranks uniformly bounded. However, this might cause a loss of positive definiteness, which has been discussed in [13]. On the lowest level of recursions in (2.25) and (2.27) standard dense Cholesky decompositions are computed, contributing with the cost $\mathcal{O}(n_{\min}^3)$. This leads to Algorithm 2.6.

Hence, the computational cost of an approximate Cholesky decomposition in the HODLR format satisfies

$$\mathcal{C}_{\text{hchol}}(n) = \begin{cases} \mathcal{O}(n_{\min}^3), & \text{if } n = n_{\min}\ (p = 0), \\ 2\mathcal{C}_{\text{hchol}}\left(\frac{n}{2}\right) + \mathcal{O}(k^3 n + k^2 n \log n), & \text{otherwise.} \end{cases}$$

Applying the Master theorem to the previous recurrence yields

$$\mathcal{C}_{\text{hchol}}(n) \in \mathcal{O}(k^3 n \log n + k^2 n \log^2 n). \tag{2.28}$$

Algorithm 2.6 Approximate Cholesky decomposition

Input: Symmetric positive definite $M \in \mathcal{H}_{n\times n}(k)$ of level p.
Output: Upper triangular $R \in \mathcal{H}_{n\times n}(k)$ of level p such that $M \approx R^T R$.

1: **function** $R = \texttt{hchol}(M)$
2: **if** $p = 0$ **then**
3: Return $R = \texttt{chol}(M)$.
4: **else**
5: Partition $M = \begin{bmatrix} M_{11} & U_1 V_2^T \\ V_2 U_1^T & M_{22} \end{bmatrix}$.
6: Call recursively $R_{11} = \texttt{hchol}(M_{11})$.
7: Solve $R_{11}^T X_1 = U_1$ for X_1 by forward substitution.
8: Set $Y_2 = V_2$.
9: Compute $S = M_{22} -_{\mathcal{H}} Y_2 X_1^T X_1 Y_2^T$.
10: Call recursively $R_{22} = \texttt{hchol}(S)$.
11: Return $R = \begin{bmatrix} R_{11} & X_1 Y_2^T \\ 0 & R_{22} \end{bmatrix}$.
12: **end if**
13: **end function**

We mention that the computation of an approximate LU decomposition in the HODLR format is derived and computed similarly [67, Chapter 3].

2.2.7 Submatrix selection

In this work, we also consider the extraction of submatrices of HODLR matrices. Let $M \in \mathcal{H}_{n\times n}(k)$, associated with an integer partition $n = n_1 + \cdots + n_{2^p}$, and consider a subset of indices $C \subset \{1, \ldots, n\}$. Then the submatrix $M(C, C)$ is again a HODLR matrix. To see this, consider the partitioning (2.1) and let $C = C_1 \cup C_2$ with $C_1 = C \cap [1, n_1^{(1)}]$ and $C_2 = C \cap [n_1^{(1)} + 1, n]$, where $n_1^{(1)}$ is the size of $M_{11}^{(1)}$. Then

$$\begin{aligned} M(C,C) &= \left[\begin{array}{c|c} M_{11}^{(1)}(C_1,C_1) & M_{12}^{(1)}(C_1,C_2) \\ \hline M_{21}^{(1)}(C_2,C_1) & M_{22}^{(1)}(C_2,C_2) \end{array}\right] \\ &= \left[\begin{array}{c|c} M_{11}^{(1)}(C_1,C_1) & U_1^{(2)}(C_1,:)\Big(V_2^{(2)}(C_2,:)\Big)^T \\ \hline U_2^{(2)}(C_2,:)\Big(V_1^{(2)}(C_1,:)\Big)^T & M_{22}^{(1)}(C_2,C_2) \end{array}\right]. \end{aligned}$$

Hence, the off-diagonal blocks again have rank at most k. Applying this argument recursively to $M_{11}^{(1)}(C_1, C_1)$, $M_{22}^{(1)}(C_2, C_2)$ establishes $M(C, C) \in \mathcal{H}_{m\times m}(k)$, associated with the integer partition

$$|C| =: m = m_1 + m_2 + \cdots m_{2^p},$$

where m_1 is the cardinality of $C \cap [1, n_1]$, m_2 is the cardinality of $C \cap [n_1 + 1, n_1 + n_2]$, and so on. Note that it may happen that some $m_j = 0$, in which case the corresponding blocks in the HODLR format vanish. Formally, this poses no problems in the definition and operations with HOLDR matrices. In practice, these blocks are removed to reduce overhead.

2.2.8 Existing HODLR QR decompositions

To our knowledge, three different algorithms [11, 17, 84] have been proposed so far for computing a QR decomposition in the hierarchical matrix format. However, each of them seems to have some drawbacks, e.g., failing to achieve a highly accurate decomposition or leading to loss of orthogonality in the orthogonal factor in typical applications. In the following we provide a review of these methods.

Additionally, we mention that in Chapter 5 we employ one of the existing algorithms to compute an orthonormal factor from the QR decomposition of a well-conditioned rectangular matrix. In particular, we use an algorithm proposed by Lintner in [84]; see the method below.

Lintner's QR decomposition

Let $M \in \mathbb{R}^{n\times m}$ with full column rank, and let $M = QR$ be the QR decomposition of M. Then the following equalities hold

$$M^T M = R^T Q^T Q R = R^T R,$$

since Q is orthogonal. It follows that the upper triangular factor R is the Cholesky factor of the positive definite matrix $M^T M$. Thus, the QR decomposition can be obtained in two steps:

1. compute the Cholesky factor R of $M^T M$,

2. solve the upper triangular system $M = QR$.

This method is also known as the Cholesky QR method [106, 107].

For a HODLR matrix $M \in \mathcal{H}_{n\times m}(k)$ of level p, steps 1 and 2 can be performed using only the operations discussed in Sections 2.2.4, 2.2.5 and 2.2.6. Therefore Lintner's method can be implemented in the HODLR format with the complexity $\mathcal{O}(k^2 n \log^2 n)$. The pseudocode is provided in Algorithm 2.7.

Algorithm 2.7 Lintner's $\mathcal{H}$QR decomposition

Input: $M \in \mathcal{H}_{n\times m}(k)$ of level p.
Output: Orthonormal $Q \in \mathcal{U}_{n\times m}(k)$, upper triangular $R \in \mathcal{U}_{m\times m}(k)$ with $M \approx QR$.

1: Compute $B = M^T *_{\mathcal{H}} M$.
2: Compute $R = \texttt{hchol}(B)$.
3: Solve the upper triangular system $M = QR$ using a variant of Algorithm 2.5.
4: Return Q and R.

The biggest disadvantage of this simple method is the squaring of the condition number when computing R. Indeed, we have that $\kappa(M^T M) \approx \kappa(M)^2$. Therefore, for a badly conditioned matrix M, this causes a loss of orthogonality in Q, and lack of accuracy. This can be easily noticed in Example 2.5 and Example 2.6.

To reduce the deviation from the orthogonality in Q, Lintner proposed to perform a reorthogonalization step as follows. Assuming that Q and R are computed as aforementioned, we have

$$A = QR = Q_1 R_1 R,$$

where $Q = Q_1 R_1$ is a QR decomposition of Q. This leads to a QR decomposition of A with the orthogonal factor Q_1 and the upper triangular factor $R_1 R$.

Indeed, as shown in [116], this approach, called the CholeskyQR2 algorithm, improves both accuracy and orthogonality if the matrix condition number is not too large. However, for hierarchical matrices, as a result of the approximative arithmetic, several reorthogonalization steps might be needed to obtain the orthogonal factor, which could potentially lead to a significant increase of the computational cost.

To cope with the squaring of the condition number arising in the previous approach, Lintner derived an alternative method which involves the computation of the polar decomposition via an iterative method similar to the sign-function iteration [75]. However, this method involves the matrix inversion in each iteration, which makes it more expensive than other approaches. Still, the asymptotic complexity is shown to be linear-polylogarithmic for a hierarchical matrix A.

In particular, Lintner suggests to perform the following steps: first compute the polar decomposition $A = QH$. Then compute the Cholesky decomposition of $H = R^T R$, followed by the Cholesky QR decomposition $R^T = Q_1 R_1$. This results in the QR decomposition of A

$$A = QH = QR^T R = (QQ_1)(R_1 R),$$

with the orthogonal factor equal to QQ_1 and the upper triangular factor to $R_1 R$. This approach is appealing since the Cholesky QR decomposition is applied to matrices with $\kappa(R) = \sqrt{\kappa(A)}$.

Bebendorf's QR decomposition

The method proposed by Bebendorf [11, Chapter 2] applies a sequence of orthogonal matrices to a hierarchical matrix in order to reduce the matrix to an upper triangular form. Let $M \in \mathbb{R}^{n \times n}$ be partitioned as

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

and M_{11} invertible. Then by setting $X = M_{12} M_{11}^{-1}$, we can compute the Cholesky decompositions of the symmetric positive definite matrices $I + X^T X$ and $I + X X^T$, denoting the corresponding Cholesky factors by R_1 and R_2, respectively. Using these Cholesky factors we define the orthogonal matrix Q as

$$Q := \begin{bmatrix} R_1^{-T} & 0 \\ 0 & R_2^{-T} \end{bmatrix} \begin{bmatrix} I & X^T \\ -X & I \end{bmatrix}.$$

In fact, the transformation Q can be considered as a block Givens rotation, since $\det(Q) = 1$. Applying Q to M from the left yields an upper triangular matrix:

$$QM = \begin{bmatrix} R_1 M_{11} & R_1^{-T}(M_{12} + X^T M_{22}) \\ 0 & R_2^{-T}(M_{22} - X M_{12}) \end{bmatrix}.$$

This procedure is then applied recursively to the diagonal blocks. For a hierarchical matrix M, the recursion stops on the lowest hierarchical level, where the standard QR decomposition is computed. All involved operations can be performed efficiently in the hierarchical matrix format. Furthermore, Bebendorf showed that this method exhibits $\mathcal{O}(k^2 n \log^2 n)$ computational cost.

We remark that in each recursion step the first diagonal block needs to be inverted, which can potentially cause problems with the execution of the method.

Benner's and Mach's QR decomposition

Benner and Mach [17] have developed a method for computing a QR decomposition of a hierarchical matrix, based on a block recursive QR decomposition [60]. The idea is to recursively split the matrix into block columns corresponding to the hierarchical partitioning, and to compute dense QR decompositions on the lowest level of subdivision. In particular, their approach is based on the following procedure. Let $M \in \mathbb{R}^{n \times n}$ have the QR decomposition $M = QR$. Partitioning M as

$$[M_1\ M_2] = [Q_1\ Q_2] \begin{bmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{bmatrix},$$

with $n_1 = \lfloor \frac{n}{2} \rfloor$, $M_1, Q_1 \in \mathbb{R}^{n \times n_1}$ and $M_2, Q_2 \in \mathbb{R}^{n \times n - n_1}$, yields three steps:

1. computing the QR decomposition $M_1 = Q_1 R_{11}$,
2. solving for R_{12} the system $Q_1^T M_2 = R_{12}$, and
3. computing the QR decomposition $M_2 - Q_1 R_{12} = Q_2 R_{22}$.

Finally, steps 1 and 3 are computed recursively, where the recursion is terminated for sufficiently small block columns. All three steps can be efficiently performed in the

hierarchical matrix arithmetic, as only matrix-matrix multiplications and additions are needed on higher levels of the hierarchical structure. On the lowest level of hierarchical structure, the authors suggest to compute QR decompositions of compressed block columns, similarly as described in Section 2.2.9 below. Moreover, the orthogonalization is performed using a block modified Gram-Schmidt procedure. The computed matrices Q and R have the same hierarchical structure as the starting matrix. Moreover, the method exhibits $\mathcal{O}(k^2 n \log^2 n)$ computational complexity.

The analysis in [17] has shown that none of the three presented methods is superior to each other. Moreover, their efficiency in terms of accuracy, and orthogonality in Q seems to be problem dependent.

2.2.9 A new method for computing HODLR QR decomposition

In this section we present a new method for computing a Householder based QR decomposition of a $n \times n$ HODLR matrix. Given $M \in \mathcal{H}_{n\times n}(k)$ of level p with the integer partition (2.2), we aim to compute its QR decomposition

$$M = (I - YTY^T)R, \tag{2.29}$$

where the orthogonal factor Q is given in terms of a compact WY representation, and matrices $Y, T, R \in \mathcal{H}_{n\times n}(k)$ have the same HODLR structure as M. The new method is applied in a recursive way on block columns, akin to the method proposed by Benner and Mach. The fundamental differences arise in the computation and in the storage of the orthogonal factor Q, likewise in the orthogonalization step. As mentioned earlier, the authors in [17] perform the orthogonalization using a block modified Gram-Schmidt procedure, which in comparison to the Householder based approach in general results with poorer orthogonality [60, Section 5.2].

In the following we start by recalling Householder reflectors, and the storage efficient WY representation of an orthogonal factor. Moreover, we describe a block recursive procedure for computing a QR decomposition of a matrix proposed in [51], which serves as a basis for our method. The derivation of the new method is proceeded in two stages: first we derive the computations on the lowest level of subdivision of HODLR structure. Here we compute a QR decomposition of a compressed sub-block column utilizing a compact WY representation. Then we describe how to process the intermediate results

to obtain the decomposition on higher levels of HODLR structure. In particular, only HODLR addition and the multiplication with low-rank matrices are required to perform the computations on higher levels. Moreover, we show that this leads to a method with a linear-polylogarithmic complexity.

Similar ideas based on compressed block columns have been pursued in the process of compression of a dense matrix into the HSS format, as well as for obtaining a fast ULV solver in the HSS format; see [33]. However, we are not aware of such methods being used for the computations within the HODLR format.

Preliminaries

An $n \times n$ *Householder reflector* takes the form

$$H = I - \beta v v^T, \quad v \in \mathbb{R}^n, \quad \beta = 2/\|v\|_2^2.$$

The vector v is the *Householder vector*. For a vector $x \in \mathbb{R}^n$ there always exists $v \in \mathbb{R}^n$ such that Hx is a scalar multiple of the first unit vector e_1, i.e. $Hx = \pm\|x\|_2 e_1$. For a matrix $M \in \mathbb{R}^{n\times m}$, its QR decomposition can therefore be obtained by applying a sequence of Householder reflectors $H_1, \ldots, H_m$ from the left to M, as shown in Algorithm 2.8. Moreover, the orthogonal factor Q is the product of m Householder reflectors $H_j, j = 1, \ldots, m$, [60, Section 5.2].

As shown in [101], a product of m Householder projectors can be compactly written as

$$H_1 H_2 \cdots H_m = I - Y T Y^T, \tag{2.30}$$

where a lower trapezoidal $Y \in \mathbb{R}^{n\times m}$, as in Algorithm 2.8, contains the corresponding Householder vectors and $T \in \mathbb{R}^{m\times m}$ is an upper triangular matrix. The factorization (2.30) is the so-called *compact WY representation*, and allows for efficiently storing m Householder reflectors, as well as for applying a sequence of Householder reflectors in terms of matrix–matrix products. The computation of a compact WY representation can be carried out as given in Algorithm 2.9.

Our method is based on a recursive block column approach for the computation of a QR decomposition using the compact WY representation, which is also suggested in [51].

Algorithm 2.8 Householder QR

Input: $M \in \mathbb{R}^{n\times m}$, with $n \geq m$.

Output: A lower trapezoidal $Y \in \mathbb{R}^{n\times m}$, with Y storing Householder reflectors generators, scalars β_j, an upper triangular $R \in \mathbb{R}^{m\times m}$ such that $M = Q \begin{bmatrix} R \\ 0 \end{bmatrix}$, with an orthogonal $Q = (I - \beta_1 v_1 v_1^T) \cdots (I - \beta_k v_m v_m^T)$.

1: Set $Y = 0_{n\times m}$.
2: **for** $j = 1, \ldots m$ **do**
3: Generate a Householder reflector $H_j = I - \beta_j v_j v_j^T$ for annihilating $M(j : n, j)$.
4: Set $Y(j : n, j) = v_j$.
5: Update $M(j : n, j : m) = H_j M(j : n, j : m)$.
6: **end for**
7: Set $R = M(1 : m, 1 : m)$.
8: Return Y, $\beta_1, \ldots, \beta_m$ and R.

Algorithm 2.9 Compact WY representation of a product of m Householder reflectors

Input: A lower trapezoidal $Y \in \mathbb{R}^{n\times m}$ containing the Householder vectors, and the associated scalars $\beta_j, j = 1, \ldots m$, generating Householder reflectors $H_1, \ldots H_m$.

Output: An upper triangular $T \in \mathbb{R}^{m\times m}$ such that $H_1 \cdots H_m = I - YTY^T$.

1: **for** $j = 1, \ldots m$ **do**
2: **if** $j = 1$ **then**
3: Set $T = \beta_1$.
4: **else**
5: Compute $z = -\beta_j TY(:, 1 : j - 1)^T Y(:, j)$.
6: Set $T = \begin{bmatrix} T & z \\ 0 & \beta_j \end{bmatrix}$.
7: **end if**
8: **end for**

For $M \in \mathbb{R}^{n\times m}$, its QR decomposition

$$M = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix} = Q \begin{bmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{bmatrix}, \tag{2.31}$$

is obtained by first recursively computing a QR decomposition of the first block column consisting of $\lfloor \frac{m}{2} \rfloor$ columns,

$$\begin{bmatrix} M_{11} \\ M_{21} \end{bmatrix} = Q_1 \begin{bmatrix} R_{11} \\ 0 \end{bmatrix}, \text{ with } Q_1 = I - Y_1 T_{11} Y_1^T. \tag{2.32}$$

The second block column of M is updated using the QR decomposition of the first block column

$$Q_1^T \begin{bmatrix} M_{21} \\ M_{22} \end{bmatrix} = \begin{bmatrix} R_{12} \\ \tilde{M}_{22} \end{bmatrix}, \tag{2.33}$$

and lastly the QR decomposition of $\tilde{M}_{22}$ is computed recursively

$$\tilde{M}_{22} = \tilde{Q}_2 R_{22}, \text{ with } \tilde{Q}_2 = I - Y_2 T_{22} Y_2^T. \tag{2.34}$$

The recursion is terminated when the number of columns in the matrix to be factorized is less than a prescribed block column size n_b.

By setting

$$Q = Q_1 \operatorname{diag}(I, \tilde{Q}_2), \quad R = \begin{bmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{bmatrix},$$

we obtain the QR decomposition (2.31) of M. Moreover, compact WY representations (2.32) and (2.34) of Q_1 and $\tilde{Q}_2$ can be used to represent Q in the same format. Indeed, by appending a zero matrix to Y_2 such that $Y_2 := \begin{bmatrix} 0 \\ Y_2 \end{bmatrix}$ has the same number of rows as M, we obtain that the orthogonal factor Q can be written as

$$Q = (I - Y_1 T_{11} Y_1^T)(I - Y_2 T_{22} Y_2^T) = I - Y T Y^T, \tag{2.35}$$

where

$$Y = \begin{bmatrix} Y_1 & Y_2 \end{bmatrix}, \quad T = \begin{bmatrix} T_{11} & -T_{11} Y_1^T Y_2 T_{22} \\ 0 & T_{22} \end{bmatrix}. \tag{2.36}$$

Therefore, the computation of the factors Y and T, representing the orthogonal factor Q, can be performed in a recursive manner, with the recursion applied on block columns. Algorithm 2.10 summarizes the described procedure, and its computational cost for an n by m matrix, with $n \geq m$, is $\mathcal{O}(m^2 n)$.

In the subsequent sections we tailor this method to the HODLR format. Furthermore, Algorithm 2.10 is used on the lowest level of the HODLR subdivision.

Algorithm 2.10 Recursive block QR decomposition

Input: Matrix $M \in \mathbb{R}^{n\times m}$, with $n \geq m$, minimal block column size n_b.

Output: QR decomposition of $M = Q\begin{bmatrix} R \\ 0 \end{bmatrix}$, where orthogonal $Q \in \mathbb{R}^{n\times n}$ is stored as $Q = I - YTY^T$, with lower trapezoidal $Y \in \mathbb{R}^{n\times m}$ and upper triangular $T, R \in \mathbb{R}^{m\times m}$.

function $[Y, T, R] =$ `blockQR`(M, n_b)
if $m \leq n_b$ **then**
  Apply Algorithm 2.8 and Algorithm 2.9 to M to compute Y, T, R.
else
  Set $m_1 = \lfloor m/2 \rfloor$.
  Compute $[Y_1, T_{11}, R_{11}] =$ `blockQR`$(M(:, 1 : m_1), n_b)$ recursively.
  Compute $M(:, m_1 + 1 : m) \leftarrow (I - Y_1 T Y_1^T)^T M(:, m_1 + 1 : m)$.
  Compute $[Y_2, T_{22}, R_{22}] =$ `blockQR`$(M(m_1 + 1 : n, m_1 + 1 : m), n_b)$ recursively.
end if
Return $Y = \begin{bmatrix} Y_1 & Y_2 \end{bmatrix}$ and $T = \begin{bmatrix} T_{11} & -T_{11}Y_1^T Y_2 T_{22} \\ 0 & T_{22} \end{bmatrix}$, with $Y_2 = \begin{bmatrix} 0 \\ Y_2 \end{bmatrix}$.
Return $R = \begin{bmatrix} R_{11} & M(1 : m_1, m_1 + 1 : m) \\ 0 & R_{22} \end{bmatrix}$.
end function

QR decomposition of a block column on the lowest HODLR level

Let $M \in \mathcal{H}_{n\times n}(k)$ of level p with the integer partition (2.2), and let $n^{(i)} = \sum_{j=1}^{i} n_j$, for $i = 1, \ldots, 2^p$ and $n^{(0)} = 0$. Thus, on the lowest HODLR level, M is divided into 2^p block columns of size $n \times n_i, i = 1, \ldots 2^p$, where each block column contains exactly one dense diagonal block $M_{ii}^{(p)}$. Let us enumerate those block columns from 1 to 2^p.

Motivated by (2.34), we consider a sub-block of the ith block column of M, i.e. $M_i := M(n^{(i-1)} + 1 : n, n^{(i-1)} + 1 : n^{(i)})$. Let b denote the number of the off-diagonal blocks that intersect with M_i, and that are below the diagonal block $M_{ii}^{(p)}$; see Figure 2.4 for an illustration. Then we obtain the following factorization

$$M_i = \begin{bmatrix} M_{ii}^{(p)} \\ U_1 V_1^T \\ \vdots \\ U_b V_b^T \end{bmatrix} = \operatorname{diag}(I, Q_1, \ldots, Q_b) \begin{bmatrix} M_{ii}^{(p)} \\ R_1 V_1^T \\ \vdots \\ R_b V_b^T \end{bmatrix} =: Q^{(i)} \hat{V} = Q^{(i)} \begin{bmatrix} \square \\ 1 \\ \vdots \\ b \end{bmatrix}, \quad (2.37)$$

where $U_j = Q_j R_j$ is a QR decomposition of $U_j, j = 1, \ldots, b$. For the off-diagonal

blocks of with the column size larger than n_i, matrices V_j denote the restrictions of the corresponding right low-rank factors $V^{(l)}$ on the block column, as shown in Figure 2.4. QR decompositions of the factors U_j are computed only the first time an off-diagonal block contributes to a block column, and they are reused for the computations in the subsequent block columns. Moreover, factors $V^{(l)}$ are multiplied from the left by R_j the first time the corresponding off-diagonal block intersects a block column.

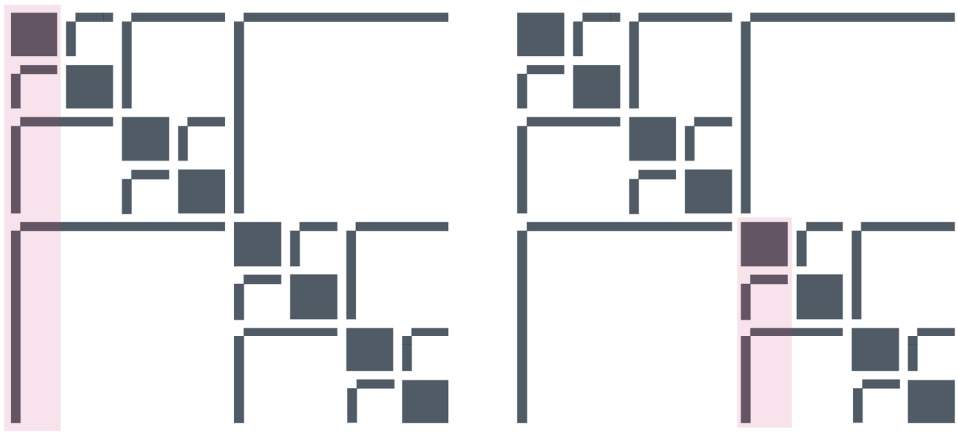

Figure 2.4 – Left: Highlighted block column is considered in (2.37) *for* $i = 1$. *Right: Highlighted block column is considered in* (2.37) *for* $i = 5$.

The computation of the QR decomposition in (2.37) is proceeded by obtaining a Householder QR decomposition of the compressed ith block column, i.e. dense matrix $\hat{V} \in \mathbb{R}^{(n_i+bk)\times n_i}$, and by storing its orthogonal factor in the compact WY form. In particular, we get

$$\hat{V} = (I - \hat{Y}\hat{T}\hat{Y}^T) \begin{bmatrix} \hat{R} \\ 0 \end{bmatrix}. \tag{2.38}$$

Incorporating decomposition (2.38) into decomposition (2.37), and subdividing $\hat{Y}$ accordingly to the blocks in $\hat{V}$, we obtain a QR decomposition with the orthogonal factor Q in

the compact WY form

$$M(n^{(i-1)}+1:n, n^{(i-1)}+1:n^{(i)}) = Q^{(i)}(I - \hat{Y}\hat{T}\hat{Y}^T)\begin{bmatrix}\hat{R}\\0\end{bmatrix}$$
$$= \left(I - \begin{bmatrix}\hat{Y}_{ii}\\Q_1\hat{Y}_1\\\vdots\\Q_b\hat{Y}_b\end{bmatrix}\hat{T}\begin{bmatrix}\hat{Y}_{ii}\\Q_1\hat{Y}_1\\\vdots\\Q_b\hat{Y}_b\end{bmatrix}^T\right)\begin{bmatrix}\hat{R}\\0\end{bmatrix} =: (I - Y_i\hat{T}Y_i^T)\begin{bmatrix}\hat{R}\\0\end{bmatrix}. \tag{2.39}$$

The orthogonality in (2.39) is verified by a straightforward computation.

We note that Y_i is the ith sub-block column of a lower triangular HODLR matrix Y from (2.29). In particular, matrix $\hat{Y}_{ii}$ is the dense diagonal block in the ith block column of Y, while matrices $Q_j, \hat{Y}_j, j = 1, \ldots, b$ are low-rank factors of the corresponding off-diagonal blocks. Moreover, matrices $\hat{R}$ and $\hat{T}$ are dense diagonal blocks in the ith block column of matrices R and T in (2.29), respectively.

Moreover, we remark that for a block column enumerated with an odd number, we always need to perform at least one QR decomposition of a left low-rank factor of an off-diagonal block, while for a block column enumerated with an even number this operation is never performed. In fact, since this procedure is applied to all block columns on the lowest HODLR level, QR decompositions of all left low-rank factors in the lower off-diagonal blocks ought to be computed.

The procedure described above is summarized in Algorithm 2.11. This algorithm requires only standard matrix-matrix multiplications and standard QR decompositions, implying that the accuracy of the computed decomposition is inherited from the standard operations.

The computational complexity of Algorithm 2.11 depends on the number of QR decompositions computed in Line 2. The maximal computational cost occurs in the first block column of a HODLR matrix, when QR decompositions of all off-diagonal blocks intersecting the block column need to be computed. Its impact on the computational complexity of the overall algorithm is provided later on.

In the following, we illustrate the computations in the second block column on the lowest level of the HODLR format, as this will be insightful for developing and understanding a

general scheme for computing HODLR QR decomposition.

Algorithm 2.11 QR decomposition of a block column on the lowest HODLR level

Input: Block column $M_i = M(n^{(i-1)}+1 : n, n^{(i-1)}+1 : n^{(i)})$ as in (2.37) of $M \in \mathcal{H}_{n\times n}(k)$, minimal block column size n_b.

Output: QR decomposition of M_i, $M_i = (I - Y_i\hat{T}Y_i^T)\begin{bmatrix}\hat{R}\\0\end{bmatrix}$, as in (2.39).

1: **for** $j = 1, \ldots, b$ **do**
2: Compute a thin QR decomposition $U_j = Q_j R_j$ (if needed).
3: **end for**
4: Assemble $\hat{V}^T = \begin{bmatrix}(M_{ii}^{(p)})^T & V_1R_1^T & \cdots & V_bR_b^T\end{bmatrix}$.
5: Compute $[\hat{Y}, \hat{T}, \hat{R}] = \texttt{blockQR}(\hat{V}, n_b)$.
6: Return $Q_j, j = 1, \ldots, b$.
7: Return $\hat{Y}, \hat{T}, \hat{R}$.

Illustration of the update in the second block column

Suppose the QR decomposition of the first block column of M on the lowest HODLR level had been obtained by the procedure explained above, yielding the representation (2.39). This implies that the first block columns of Y, T and R in (2.29) had been computed as well.

Analogously to (2.33), we proceed by updating the second block column of M by multiplying it from the left with $(I - Y_1\hat{T}^TY_1^T)$, computed in (2.39). Taking into account the computations for the off-diagonal blocks in the first block column of M in (2.37), the second block column can be written as

$$\tilde{M}_2 = \begin{bmatrix} U_1^{(p)}(V_2^{(p)})^T \\ M_{22}^{(p)} \\ Q_2R_2V \\ \vdots \\ Q_bR_bV \end{bmatrix} = \operatorname{diag}(I, I, Q_2, \ldots, Q_b) \begin{bmatrix} U_1^{(p)}(V_2^{(p)})^T \\ M_{22}^{(p)} \\ R_2\tilde{V}_2 \\ \vdots \\ R_b\tilde{V}_b \end{bmatrix}. \tag{2.40}$$

The update $(I - Y_1\hat{T}Y_1^T)^T\tilde{M}_2$ can therefore be performed in an inexpensive manner by exploiting the structure of blocks in (2.39) and (2.40). In particular, the representation

of the update is as follows

$$(I - Y_1\hat{T}^T Y_1^T)\tilde{M}_2 = \operatorname{diag}(I, I, Q_2, \ldots, Q_b)\left(I - \begin{bmatrix} \hat{Y}_{ii} \\ Q_1\hat{Y}_1 \\ \hat{Y}_2 \\ \vdots \\ \hat{Y}_b \end{bmatrix} \hat{T}^T \begin{bmatrix} \hat{Y}_{ii} \\ Q_1\hat{Y}_1 \\ \hat{Y}_2 \\ \vdots \\ \hat{Y}_b \end{bmatrix}^T\right)\begin{bmatrix} U_1^{(p)}(V_2^{(p)})^T \\ M_{22}^{(p)} \\ R_2\tilde{V}_2 \\ \vdots \\ R_b\tilde{V}_b \end{bmatrix}. \tag{2.41}$$

Hence, to perform the update (2.41), operations on the compressed first block column of Y and the compressed second block column of M are employed. In fact, the computation in (2.41) does not involve the multiplication with $\operatorname{diag}(I, I, Q_2, \ldots, Q_b)$, i.e. only the multiplication of the second part is performed, as illustrated in Figure 2.5. In order to carry out the calculation the low-rank matrix arithmetic is used.

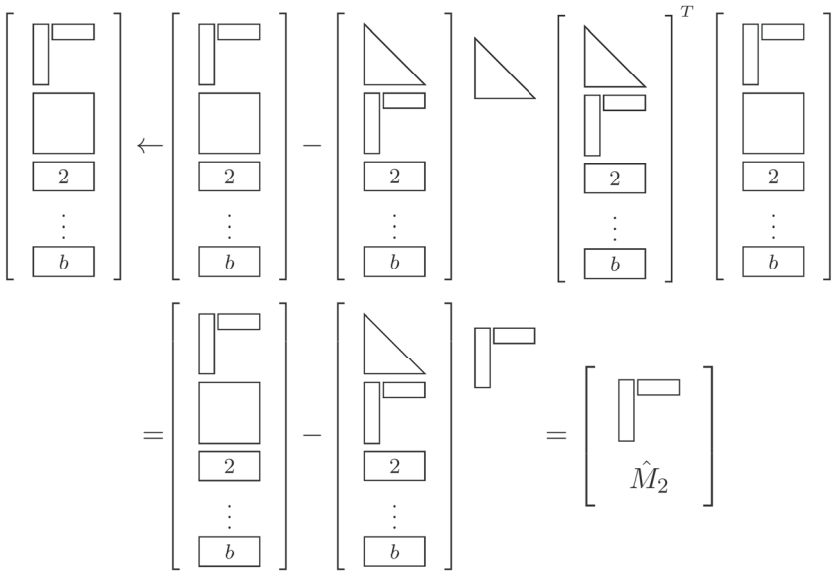

Figure 2.5 – Illustration of the update of the second block column on the lowest IIODLR level.

Thus, from the computations shown in Figure 2.5, we obtain a factorization of (2.41):

$$(I - Y_1\hat{T}^T Y_1^T)\tilde{M}_2 = \operatorname{diag}(I, I, Q_2, \ldots, Q_b)\begin{bmatrix} \tilde{U}_1^{(p)}(\tilde{V}_2^{(p)})^T \\ \hat{M}_2 \end{bmatrix}. \tag{2.42}$$

The computation of a QR decomposition of the second block column is completed after applying Algorithm 2.11 to $M_2 = \mathrm{diag}(I, Q_2, \ldots, Q_b)\hat{M}_2$.

QR decomposition for HODLR matrices

Having described a method for computing a structured QR decomposition of a block column on the lowest level of HODLR structure, and the computation of the update in the second block column, we derive a QR decomposition of a complete HODLR matrix, based on a recursive block column approach.

For a HODLR matrix $M \in \mathcal{H}_{n\times n}(k)$, its QR decomposition in the HODLR format is given in terms of triangular matrices $Y, T, R \in \mathcal{H}_{n\times n}(k)$, as given in (2.29). Our method for computing a HODLR QR decomposition therefore consists of the following steps:

1. the computation of a QR decomposition of the first block column recursively until the lowest HODLR level had been reached;

2. the computation of the update in the second block column;

3. the computation of a QR decomposition of the updated second block column (starting from a diagonal block), recursively until the lowest HODLR level had been reached.

In particular, for a HODLR matrix M, based on its HODLR partitioning

$$M = \begin{bmatrix} M_{11} & U_1 V_2^T \\ U_2 V_1^T & M_{22} \end{bmatrix},$$

we decompose M into two block columns $M = \begin{bmatrix} M_1 & \tilde{M}_2 \end{bmatrix}$, with

$$M_1 = \begin{bmatrix} M_{11} \\ U_2 V_1^T \end{bmatrix}, \quad \tilde{M}_2 = \begin{bmatrix} U_1 V_2^T \\ M_{22} \end{bmatrix}.$$

Firstly, the computation of a QR decomposition of M_1 is carried out recursively in the HODLR arithmetics, yielding $M_1 = Q_1 *_{\mathcal{H}} R_1$, with Q_1 stored in terms of the compact WY representation. After updating the second block column by exploiting the structure

of Q_1, as in Figure 2.5:

$$\begin{bmatrix} \tilde{U}_1 \tilde{V}_2^T \\ M_2 \end{bmatrix} = Q_1^T *_{\mathcal{H}} \tilde{M}_2,$$

we compute recursively a QR decomposition of M_2, with the orthogonal factor given in the compact WY representation. Moreover, the computation of the upper off-diagonal block of T is performed as in (2.36) by using the HODLR arithmetic and taking advantage of the structure of the block columns of a lower triangular matrix. Analogously to (2.35), this method yields a QR decomposition of M and the pseudocode is provided in Algorithm 2.12.

Algorithm 2.12 Householder QR decomposition of a HODLR matrix (hQR)

Input: A HODLR matrix M or a block column M of a HODLR matrix, minimal block column size n_b, truncation tolerance ϵ.
Output: HODLR QR decomposition (2.29) of M.

1: **function** $[Y, T, R] = \texttt{hQR}(M, n_b, \epsilon)$
2: **if** on the lowest HODLR level **then**
3: Return Y, T, R by applying Algorithm 2.11 to M.
4: **else**
5: Set $[n, m] = \texttt{size}(M)$.
6: Partition M into two block columns $M = \begin{bmatrix} M_1 & \tilde{M}_2 \end{bmatrix}$, with M_1 of size $n \times n_1$ and $\tilde{M}_2$ of size $n \times m - n_1$, according to the HODLR partitioning.
7: Compute $[Y_1, T_{11}, R_{11}] = \texttt{hQR}(M_1, n_b, \epsilon)$, with $Q_1 = I - Y_1 T_1 Y_1^T$.
8: Update $\tilde{M}_2 = Q_1^T *_{\mathcal{H}} \tilde{M}_2$ (see Figure 2.5).
9: Compute $[Y_2, T_{22}, R_{22}] = \texttt{hQR}(\tilde{M}_2(n_1 + 1 : n, :), n_b, \epsilon)$.
10: Compute $T_{12} = -T_{11} *_{\mathcal{H}} (Y_1^T *_{\mathcal{H}} Y_2) *_{\mathcal{H}} T_{22}$, with $Y_2 \leftarrow \begin{bmatrix} 0_{n_1 \times m - n_1} \\ Y_2 \end{bmatrix}$.
11: **end if**
12: Return $Y = \begin{bmatrix} Y_1 & Y_2 \end{bmatrix}$, $T = \begin{bmatrix} T_{11} & T_{12} \\ 0 & T_{22} \end{bmatrix}$, and $R = \begin{bmatrix} R_{11} & \tilde{M}_2(1 : n_1, 1 : n_1) \\ 0 & R_{22} \end{bmatrix}$.
13: **end function**

Complexity analysis

In the following we derive the complexity of Algorithm 2.12. To simplify the computations we assume that $n = 2^p n_{\min}$ and thus $n_i = n_{\min}$, for $i = 1, \ldots, 2^p$ in the integer partition associated with M. Moreover, we work with the assumption that all off-diagonal ranks seen in the process are bounded by a constant k.

The computational cost on the lowest HODLR level arises from:

i) QR decompositions computed for all left factors in the lower off-diagonal blocks, and updating all right factors by multiplying with upper triangular R factors,

ii) compact WY representations of compressed block columns.

We first discuss the flop count for i). On level $l \geq 1$ of HODLR subdivision there are 2^{l-1} lower off-diagonal blocks and their left and right low-rank factors are of size $2^{p-l}n_{\min} \times k$ and $k \times 2^{p-l}n_{\min}$, respectively. The operations performed in one off-diagonal block on level l are $\mathcal{O}(2^{p-l+1}n_{\min}k^2)$, where we consider the computation of the QR decomposition of the left factor, and the update of the right factor. Thus summing over the cost all levels of subdivision, we obtain

$$\sum_{l=1}^{p} \mathcal{O}(2^{l-1}2^{p-l+1}n_{\min}k^2) \in \mathcal{O}(k^2 n \log n).$$

Now we focus on ii). A HODLR matrix of level p has 2^p block columns on the lowest level of subdivision. Compressed block columns are of size $n_{\min} + b_i k \times n_{\min}$, where b_i is the number of the lower off-diagonal blocks intersecting ith block column. Then using that $b_i \leq p$, the cost of computing compact WY representations on the lowest HODLR level is

$$\sum_{i=1}^{2^p} \mathcal{O}((n_{\min} + b_i k)n_{\min}^2) \leq \mathcal{O}(2^p(n_{\min}^3 + pkn_{\min}^2)) \in \mathcal{O}(kn \log n).$$

Hence, we get that the total number of flops required in Line 3, which uses standard QR decompositions, is $\mathcal{O}(k^2 n \log n)$.

On higher HODLR levels we perform the update in Line 8, and compute the upper off-diagonal block in T in Line 10. These operations are repeated $\log n$ times. Because of the structure of the block columns of HODLR matrices involved, we only require the low-rank matrix arithmetics and the addition of a HODLR matrix with a low rank matrix to carry out these operations. The most expensive step occurs when working with two block columns of size $n/2$, with the cost $\mathcal{O}(k^3 n + k^2 n \log n)$. Therefore, we obtain that the total cost of these operations adds up to $\mathcal{O}(k^3 n \log n + k^2 n \log^2 n)$.

Summing up all derived costs, for the computational cost of an approximate QR decomposition in the HODLR format computed by Algorithm 2.12 it holds

$$\mathcal{C}_{\texttt{hqr}}(n) \in \mathcal{O}(k^3 n \log n + k^2 n \log^2 n). \tag{2.43}$$

Numerical results

In this section we demonstrate the efficiency of our method on several examples. Further examples and applications of our new method are presented in Chapter 5, in the context of a spectral divide-and-conquer method. The computations were performed in MATLAB version R2016b on a machine with the dual Intel Core i7-5600U 2.60GHz CPU, 256 KByte of level 2 cache and 12 GByte of RAM, using a single core. We note that this is a preliminary implementation, which can be optimized further by, e.g., implementing some steps in C using the MEX-function capabilities of MATLAB with direct calls to BLAS routines.

To assess the performance of our method, we examine the accuracy of the computed QR decomposition, as well as the orthogonality of Q. To this end, we consider two error measures:

$$e_{\text{orth}} = \|Q^T Q - I\|_2, \text{ with } Q = I - YTY^T, \tag{2.44}$$

$$e_{\text{acc}} = \|QR - M\|_2, \tag{2.45}$$

where M is a given HODLR matrix. In the following, we compute the errors for the new hQR algorithm and Lintner's algorithm, denoting the errors obtained with superscripts hQR and L, respectively.

Example 2.5 (Performance versus n). We first investigate the performance of our method on randomly generated HODLR matrices. We construct the HODLR structure given the minimal block size $n_{\min} = 250$ and fill the dense blocks with random matrices of the corresponding size, and the off-diagonal blocks with random low-rank factors of rank 1. We set the truncation tolerance to $\epsilon = 10^{-10}$, and the minimal block column size needed for Algorithm 2.11 to $n_b = 32$. In Figure 2.6 we present the computational times of the hQR algorithm, and Lintner's method. The new method nicely matches the $\mathcal{O}(k^2 n \log^2 n)$ reference line. Moreover, compared to a much simpler Lintner's method, our method is approximately only two times slower.

Furthermore, Table 2.3 demonstrates that the new hQR algorithm is robust with respect to the increasing condition number, and provides good results both in terms of accuracy and orthogonality. Moreover, it confirms that the orthogonality in Lintner's algorithm deteriorates as the condition number of a matrix increases.

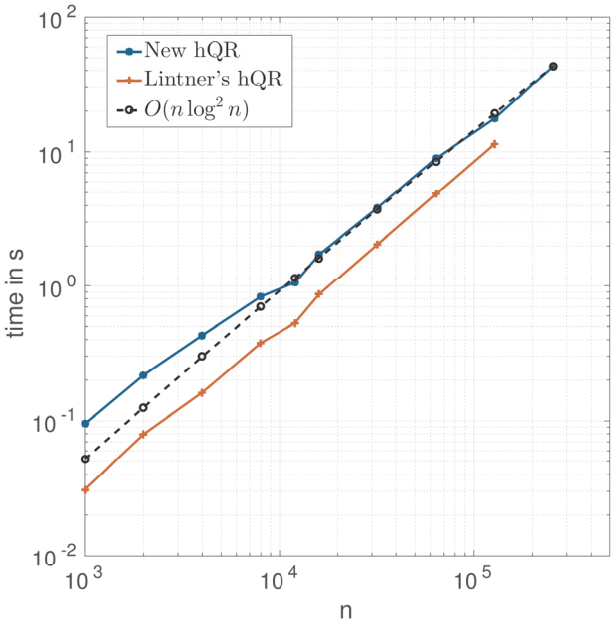

Figure 2.6 – Example 2.5. The computational time of the hQR algorithm 2.12, and Lintner's algorithm 2.7 applied to randomly generated HODLR matrices with off-diagonal rank 1*, with respect to* n*.*

Table 2.3 – Example 2.5. Accuracy and orthogonality with respect to the matrix condition number in the hQR algorithm and Lintner's algorithm applied to randomly generated HODLR matrices with off-diagonal rank 1*.*

n	$\kappa(M)$	e^{hQR}_{orth}	e^{L}_{orth}	e^{hQR}_{acc}	e^{L}_{acc}
1 000	$1.7 \cdot 10^5$	$6.8 \cdot 10^{-14}$	$6.4 \cdot 10^{-9}$	$8.7 \cdot 10^{-13}$	$2 \cdot 10^{-13}$
2 000	$3.9 \cdot 10^5$	$1.2 \cdot 10^{-13}$	$3.6 \cdot 10^{-8}$	$2.1 \cdot 10^{-12}$	$1.4 \cdot 10^{-12}$
4 000	$6.9 \cdot 10^6$	$4.1 \cdot 10^{-12}$	$8.4 \cdot 10^{-7}$	$1.1 \cdot 10^{-10}$	$2.6 \cdot 10^{-11}$
8 000	$9.8 \cdot 10^6$	$1.9 \cdot 10^{-12}$	$5.1 \cdot 10^{-6}$	$1.9 \cdot 10^{-10}$	$9.1 \cdot 10^{-10}$
12 000	$4.5 \cdot 10^9$	$1.1 \cdot 10^{-11}$	$6.2 \cdot 10^{-3}$	$5.8 \cdot 10^{-10}$	$4.6 \cdot 10^{-10}$

Additionally, we examine the numerical off-diagonal ranks in the computed decomposition. Table 2.4 shows that the off-diagonal ranks in the computed matrices Y, T are smaller than the off-diagonal ranks in R by factor of 2, and that rank seems to grow logarithmically with n. Howeover, in the following example we show that for matrices with a stronger underlying structure, the ranks in Y and T remain bounded.

Table 2.4 – Example 2.5. HODLR ranks in the factors Y, T and R from the QR decomposition.

n	rank in Y	rank in T	rank in R
1 000	2	2	4
8 000	5	5	10
64 000	8	8	16
256 000	10	10	20

Example 2.6 (Accuracy and orthogonality for Cauchy matrices). In this example we consider Cauchy matrices of size $n = 2000$, where

- $(M_1)_{ij} = 1/(x_i - y_j)$, with $x = (x_i), y = (y_j)$ vectors of points in the intervals $[-1.25, 998.25]$ and $[-0.7, 998.9]$, respectively.

- $(M_2)_{ij} = 1/(x_i - y_j)$, with $x = (x_i), y = (y_j)$ vectors of points in the intervals $[-1.25, 998.25]$ and $[-0.45, 999.15]$, respectively.

- $(M_3)_{ij} = 1/(x_i - y_j)$, with $x = (x_i), y = (y_j)$ vectors of points in the intervals $[-1.25, 998.25]$ and $[-0.15, 999.45]$, respectively.

The intervals are chosen in this way in order to obtain matrices with full rank, and to control the condition number. In all three cases, x and y are vectors of equally spaced points perturbed with $\pm 2 \cdot 10^{-2}$, where the sign is chosen at random. The matrices are represented in the HODLR format using the truncation tolerance $\epsilon = 10^{-10}$ for compressing the off-diagonal blocks, and the minimal block size $n_{\min} = 250$. The computed HODLR representations have maximal off-diagonal ranks 22. The minimal block column size is set to $n_b = 32$.

Table 2.5 reports the obtained results, and shows that the new method exhibits satisfactory orthogonality and accuracy results. We note that Lintner's method fails to carry out the computations for a matrix with a condition number of order 10^{12}, due to the loss of positive-definiteness. Furthermore, the numerical off-diagonal ranks in matrices Y and T are lower than in R, with maximal off-diagonal ranks $22, 20$ and 35 in Y, T and R respectively.

Table 2.5 – Accuracy and orthogonality in the hQR algorithm and Lintner's method for Cauchy matrices with varying condition number.

	$\kappa(M_i)$	e^{hQR}_{orth}	e^{L}_{orth}	e^{hQR}_{acc}	e^{L}_{acc}
M_1	$4.8 \cdot 10^5$	$2.5 \cdot 10^{-10}$	$1.5 \cdot 10^{-7}$	$9.7 \cdot 10^{-10}$	$1.4 \cdot 10^{-10}$
M_2	$1.3 \cdot 10^8$	$3.9 \cdot 10^{-10}$	$1.3 \cdot 10^{-1}$	$2.3 \cdot 10^{-9}$	$9.5 \cdot 10^{-10}$
M_3	$2.9 \cdot 10^{12}$	$3.2 \cdot 10^{-10}$	-	$1.7 \cdot 10^{-9}$	-

Conclusion

In this section, we have derived a new fast method for computing a QR decomposition in the HODLR format. The new method is based on a recursive block column approach, and computes the orthogonal factor in terms of a compact WY representation. Numerical experiments reveal the efficiency of the method both in terms of accuracy and orthogonality, even for badly conditioned matrices.

2.2.10 Summary of HODLR arithmetics

Having derived arithmetic operations in the HODLR format needed in this work, for completeness we conclude this section by giving a summary of these operations. Therefore, we include Table 2.6 which summarizes their computational complexities. The operations listed in Table 2.6 with subscript $\mathcal{H}$ employ the recompression in the off-diagonal blocks in order to limit the increase of off-diagonal ranks. We note again that the recompression is done adaptively, such that the 2-norm approximation error in each off-diagonal block is bounded by a prescribed truncation tolerance ϵ.

Table 2.6 – Complexity of some arithmetic operations involving HODLR matrices: $M \in \mathcal{H}_{n\times n}(k)$ symmetric positive definite, $T \in \mathcal{H}_{m\times m}(k)$ triangular and invertible, $M_1, M_2 \in \mathcal{H}_{n\times m}(k)$, $M_3 \in \mathcal{H}_{m\times p}(k)$, $B \in \mathbb{R}^{m\times p}$, $v \in \mathbb{R}^m$.

Operation	Computational complexity
Matrix-vector multiplication: $M_1 v$	$\mathcal{O}(k\tilde{n}\log\tilde{n})$, with $\tilde{n} = \max\{n, m\}$
Matrix addition: $M_1 +_{\mathcal{H}} M_2$	$\mathcal{O}(k^2\tilde{n}\log\tilde{n})$, with $\tilde{n} = \max\{n, m\}$
Matrix-matrix multiplication: $M_2 *_{\mathcal{H}} M_3$	$\mathcal{O}(k^2\tilde{m}\log^2\tilde{m})$, with $\tilde{m} = \max\{n, m, p\}$
Cholesky decomposition: $\texttt{hchol}(M)$	$\mathcal{O}(k^2 n\log^2 n)$
QR decomposition : $\texttt{hQR}(M_1)$	$\mathcal{O}(k^2 n\log^2 n)$
Solution of triangular system: $T^{-1}B$	$\mathcal{O}(km\log m)$
Multiplication with (triangular)$^{-1}$: $M_1 *_{\mathcal{H}} T^{-1}$	$\mathcal{O}(k^2\tilde{n}\log^2\tilde{n})$, with $\tilde{n} = \max\{n, m\}$

2.2.11 MATLAB package for HODLR arithmetics

We have developed a MATLAB package that contains the implementations of the algorithms introduced in this chapter for matrices stored in the HODLR format. The package is freely available at https://anchp.epfl.ch/hodlr. In the following we give a brief overview of the data structures used and functionalities provided in the package.

The core object is a HODLR matrix implemented as a structure `hmatrix`. First we discuss the construction of matrices in the HODLR format.

Generating HODLR structure. As mentioned in Remark 2.1, we construct a specific HODLR format based on a prescribed minimal block-size $n_{\min}$. For a given matrix of size $n \times m$, we perform a matrix subdivision into four sub-blocks, such that the block $(1,1)$ is of size $\lfloor \frac{n}{2} \rfloor \times \lfloor \frac{m}{2} \rfloor$, while the other block sizes are obtained accordingly. The off-diagonal blocks $(1,2)$ and $(2,1)$ are not further subdivided, whereas the diagonal blocks $(1,1)$ and $(2,2)$ are recursively partitioned into four sub-blocks as long as their size is larger than $n_{\min}$. The off-diagonal blocks are represented in terms of their low-rank factors, and the diagonal blocks on the lowest level of recursion, i.e. on the lowest HODLR level, are stored as dense matrices.

Let b denote the total number of blocks obtained by dividing a matrix in the described manner. Then a HODLR matrix represented as a structure `hmatrix` is given in terms of the following data types:

- *blocktype*, an array of size b such that

$$blocktype(i) = \begin{cases} 1, & \text{if the } i\text{-th block is a diagonal block further subdivided,} \\ 2, & \text{if the } i\text{-th block is an off-diagonal block,} \\ 3, & \text{if the } i\text{-th block is a diagonal block on the lowest HODLR level;} \end{cases}$$

- *children*, a $2 \times 2 \times (b - l)$ tensor, where l is the number of diagonal blocks on the lowest HODLR level. For the i-th block, each component of the 2×2 matrix $children(:,:,i)$ enumerates one of its four sub-blocks. For an off-diagonal block i, $children(:,:,i)$ is the 2×2 zero matrix.

- *ind* is a $b \times 4$ matrix, where $ind(i,:)$ contains the row and column indices of the i-th

block in the original matrix. The first two components of $ind(i,:)$ are row indices of the block and the second two are column indices.

- U is a cell array of size b that stores the diagonal blocks:

$$U_i = \begin{cases} [\,], & \text{if the } i\text{-th block is further subdivided,} \\ \text{dense matrix}, & \text{otherwise.} \end{cases}$$

- C and D are cell arrays of size $b-1$ that store the low-rank factors of the off-diagonal blocks:

$$C_i, D_i = \begin{cases} \text{low rank matrix}, & \text{if the } i\text{-th block is an off-diagonal block,} \\ [\,], & \text{otherwise.} \end{cases}$$

  Moreover, an off-diagonal block i is represented as a product $C_i D_i$, i.e. C stores the left low-rank factors and D the right low-rank factors.

We now demonstrate the computation of a HODLR approximation for the matrix M_3 from Example 2.6, which is obtained by using the function `full2hmatrix`. We use the parameters $n_{\min} = 250$ and $\epsilon = 10^{-10}$, as in Example 2.6. The computed approximation is a HODLR matrix, stored as a structure H shown below.

```
>> nmin = 250;
>> epsilon = 1e-10;
>> H = full2hmatrix(M_3, nmin, epsilon);
>> H
>>    blocktype: [1 1 2 2 1 1 2 2 1 3 2 2 3 3 2 2 3 1 2 2 1 3
                  2 2 3 3 2 2 3]
>>     children: [2x2x21 double]
>>          ind: [29x4 double]
>>            U: {1x29 cell}
>>            C: {1x28 cell}
>>            D: {1x28 cell}
```

For instance, to see the enumeration of the four sub-blocks of the original matrix, we write

```
>> H.children(:,:, 1)
                          2   4
                          3   5
```

This shows that the $(1,1)$ diagonal block on the first level of the subdivision is enumerated by 2. In order to get its row and column indices with respect to M_3, we type $H.ind(2,:)$.

```
>> H.ind(2,:)
>>              1 1000 1 1000
```

In order to check the off-diagonal ranks in H, it suffices to type $H.C$. This lists all left low-rank factors of the off-diagonal blocks, as shown below.

```
>> H.C =
  Columns 1 through 11
  [] [] [1000x22 double] [1000x19 double] [] [] [500x20
    double] [500x18 double] [] [] [250x18 double]
  Columns 12 through 22
  [250x16 double] [] [] [250x18 double] [250x16 double] [] []
     [500x20 double] [500x18 double] [] []
  Columns 23 through 28
  [250x18 double] [250x16 double] [] [] [250x18 double]
    [250x16 double]
```

Moreover, to verify the accuracy of the computed HODLR approximation, we convert H into a dense matrix via the function `h2full`, and can compute the following difference in the matrix 2-norm.

```
>> norm(M_3 - h2full(H))
>>                          8.7944e-11
```

HODLR arithmetics. The operations discussed in the previous sections have been implemented taking into account the data structure `hmatrix`. A list of functions performing the aforementioned operations is given in Table 2.7.

We now demonstrate several functionalities of our package via simple examples.

The `hsum` function implements the addition of two HODLR matrices H_1 and H_2, and computes

$$H_1 \leftarrow aH_1 + bH_2, \quad a, b \in \mathbb{R},$$

for given scalars a and b. A MATLAB call to the function is shown below. The third argument is the label of blocks in H_1 and H_2 whose sum is computed.

```
>> H_1  = ... % set HODLR H_1
>> H_2  = ... % set HODLR H_2
>> epsilon =  ... % set epsilon
>> a = ... % set a
>> b = ... % set b
>> H_1 = hsum(H_1,H_2,1,epsilon,a,b);
```

Matrix-matrix multiplication is implemented in the `hmultiplication` function. For HODLR matrices H_1 and H_2, and a scalar $a \in \mathbb{R}$, the function computes

$$H \leftarrow H + aH_1H_2.$$

The corresponding MATLAB code is as follows.

```
>> H = ... % set HODLR H
>> H_1  = ... % set HODLR H_1
>> H_2  = ... % set HODLR H_2
>> epsilon =  ... % set epsilon
>> a = ... % set a
>> H = hmultiplication(H_1,H_2,H,1,1,1,epsilon,a);
```

Alongside the standard arithmetic operations, we have also implemented algorithms for the Cholesky decomposition and a QR decomposition.

For symmetric positive definite (SPD) matrices, the HODLR Cholesky decomposition is implemented in the `hchol` function. For an SPD matrix H, the function returns an upper triangular factor R such that

$$H = R^T R.$$

The following code shows how this can be done in our MATLAB package.

```
>> H = ... % set HODLR H
>> epsilon = ... % set epsilon
>> R = hmatrix(H.children,H.ind,H.blocktype,cell(l_1,1),
       cell(l_2,1),cell(l_2,1));
>> R = hchol(H,R,1,epsilon);
```

For instance, for $n = 2000$ in Example 2.5, we generate H as

```
>> n = 2000;
>> nmin = 250;
>> epsilon = 1e-10;
>> k = 1; % the off-diagonal rank
>> H = create_rand_hodlr(n,n,nmin,k);
```

With the utilities of our package, we can also easily implement Lintner's method for computing a QR decomposition given in Algorithm 2.7.

```
>> H = ... % set H
>> epsilon = ... % set epsilon
>> l_1 = length(H.U);
>> l_2 = length(H.C);
>> R = hmatrix(H.children,H.ind,H.blocktype,cell(l_1,1),
       cell(l_2,1),cell(l_2,1));
>> Q = R;
>> B = R;
>> B = hmultiplication(htranspose(H),H,B,1,1,1,epsilon,1);
>> R = hchol(B,R,1,epsilon);
>> Q = hTSolve_upper(R,H,Q,1,epsilon); % solve H = QR
```

Moreover, for H generated above, the errors in Lintner's method are displayed below.

```
>> Q = h2full(Q);
>> R = h2full(R);
>> norm(Q'*Q - eye(n))
                          3.5861e-8
>> norm(Q*R - h2full(H))
>>                        1.4032e-12
```

For more details on the functionalities of each of the functions in Table 2.7, type `help <function name>`.

Table 2.7 – A list of some functions implemented in the HODLR format.

Function	Description
`create_rand_hodlr`	Compute random HODLR matrix.
`h2full`	Convert HODLR matrix into dense matrix.
`full2hmatrix`	Convert dense matrix to HODLR matrix.
`hadd_ceye`	Add multiple of the identity matrix to HODLR matrix.
`hadd_rk`	Compute sum of HODLR matrix and low-rank matrix.
`hchol`	Compute the Cholesky decomposition of HODLR matrix.
`hextract_row_col`	Extract submatrix of HODLR matrix with given rows and colums.
`hMV`	Compute product of HODLR matrix and dense matrix.
`hmultiplication`	Compute product of two HODLR matrices.
`hscalar`	Compute scalar multiple of HODLR matrix.
`hsum`	Compute sum of two HODLR matrices.
`htrace`	Compute trace of HODLR matrix.
`htranpose`	Compute transpose of HODLR matrix.
`hTSolve_lower`	Solve lower triangular HODLR linear system with HODLR right-hand side.
`hTSolve_upper`	Solve upper triangular HODLR linear system with HODLR right-hand side.
`TSolve_lower`	Solve lower triangular HODLR linear system with a dense right-hand side.
`TSolve_upper`	Solve upper triangular HODLR linear system with a dense right-hand side.

3 Existing hierarchical methods for eigenvalue problems

In this chapter we provide a brief overview of related existing methods which address the symmetric eigenvalue problem in the context of hierarchical matrices, HODLR matrices and HSS matrices. For further details we refer to the monograph [67] and the references therein.

3.1 Projection method

The method described in the following has been proposed in [70] for computing a subset of eigenvalues contained in a specified interval.

Let $M \in \mathbb{R}^{n\times n}$ have real spectrum. Let $a, b \in \mathbb{R}$ be such that a part of the spectrum of M lies in the interval (a, b). The aim is to find a spectral projection onto $\sigma(M) \cap (a, b)$. The optimal projection is built upon the function $\chi : \mathbb{R} \to \mathbb{R}$, defined as

$$\chi(\lambda) = \begin{cases} \lambda, & \text{if } \lambda \in (a, b), \\ 0, & \text{otherwise.} \end{cases}$$

The authors in [70] propose the following rational approximation to χ

$$\tilde{\chi}(\lambda) = \frac{\lambda}{1 + T(\lambda)^{2^l}}, \text{ with } \quad T(\lambda) = \frac{2\lambda - b - a}{b - a}, l \in \mathbb{N}.$$

This is then used to obtain the projected matrix $\tilde{M}$, defined as

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

Indeed, $\tilde{M}$ is an approximation to a projection onto the invariant subspace spanned by eigenvectors associated with the eigenvalues contained in (a, b). The computation of $\tilde{M}$ requires l matrix-matrix multiplications and one matrix inversion, which can be performed efficiently in the context of hierarchical matrices. However, for large l, when $\tilde{\chi}(\lambda)$ provides a good approximation to $\chi(\lambda)$, the matrix $I + T(M)^{2^l}$ is badly conditioned. The authors suggest to overcome this problem by using a preprocessing step. Moreover, the non-zero eigenvalues of $\tilde{M}$ are the eigenvalues of M inside the interval (a, b). Their computation is performed by using the generalized Rayleigh quotient. The complexity of computing all eigenvalues is quadratic-polylogarithmic.

3.2 LR Cholesky algorithm

In this section we review a method that has been proposed in [19] for computing the spectrum of symmetric matrices.

The authors extend the LR Cholesky algorithm [98] to compute eigenvalues of symmetric HODLR matrices. Let $M \in \mathbb{R}^{n \times n}$ be symmetric positive definite. The LR Cholesky algorithm computes the iterates

$$\begin{aligned} R_{i+1}^T R_{i+1} &= M_i - \mu_i I \quad \text{(Cholesky decomposition)} \\ M_{i+1} &= R_{i+1}^{-T} M_i R_{i+1} = R_{i+1} R_{i+1}^T + \mu_i I, \end{aligned}$$

with $M_0 := M = R_0^T R_0$. The sequence of matrices M_i converges to a diagonal matrix of eigenvalues of M, while the shifts μ_i are computed so that the matrices M_i are positive definite. All operations required to perform the LR Cholesky algorithm can be efficiently computed in the HODLR arithmetic. In particular, for $M \in \mathcal{H}_{n \times n}(k)$ of level p, the iterates M_i stay within the set $\mathcal{H}_{n \times n}(pk)$. Therefore, the method exhibits $\mathcal{O}(k^2 n^2 \log^4 n)$ complexity for the computation of the complete spectrum.

The analysis provided in [19] shows that for general hierarchical matrices this method requires $\mathcal{O}(n^4)$ operations, as a result of the rank growth.

3.3 Vector iterations

In this section we review the power iteration and the preconditioned inverse iteration in the context of hierarchical matrices. The latter has been proposed in [20].

3.3.1 Power iteration

For a given matrix M and a starting vector x_0, the power iteration

$$x_{i+1} = Mx_i/\|x_i\|_2,$$

see, e.g. [100], computes the eigenvector of M related to the largest eigenvalue in the absolute sense. Because of its simple formulation, the power iteration can be performed exactly and with an almost linear cost for a hierarchical matrix M. The corresponding eigenvalue can be obtained via Rayleigh quotient

$$\mu(x) = \langle Mx, x\rangle/\|x\|_2^2.$$

For the computation of an invariant subspace of dimension l, a block version of the power iteration can be used, the so-called, simultaneous iteration.

The power iteration has already been utilized for hierarchical matrices and $\mathcal{H}^2$-matrices for estimating the 2-norm of a matrix; see [30, 29, 62, 80].

3.3.2 Inverse power iteration

When the computation of the smallest eigenvalue and/or corresponding eigenvector is required, the power iteration can be applied to M^{-1}. This results in the inverse power iteration

$$x_{i+1} = M^{-1}x_i/\|M^{-1}x_i\|_2.$$

see [100]. For obtaining eigenvalues close to a given shift $\mu \in \mathbb{R}$, the inverse power iteration can be applied to $M - \mu I$. This also improves the convergence of the inverse power iteration.

The authors in [20] have suggested to use the preconditioned inverse power iteration to compute the smallest eigenvalue and the associated eigenvector for general symmetric

hierarchical matrices. The iteration takes a form

$$x_{i+1} = x_i - T^{-1}(Mx_i - \mu(x_i)x_i),$$

where the preconditioner T is computed as the approximate inverse of M or the approximate Cholesky decomposition of M, in order to speed up the convergence. Moreover, all required operations can be performed with linear-polylogarithmic complexity for hierarchical matrices. This method has been extended for the computation of l smallest eigenvalues and eigenvectors by means of a simultaneous iteration. Additionally, the authors have shown that the computation of interior eigenvalues close to a shift μ can be carried out by applying these methods to $(M - \mu I)^2$.

The presented method requires $\mathcal{O}(k^2 n \log^2 n)$ operations to compute one eigenvalue, and it is applicable to HODLR matrices as well.

3.4 Bisection method

The method given in this section has been derived in [18] for rank-k HODLR matrices, and can be used for computing a part of the spectrum, as well as for the complete spectrum.

Let $M \in \mathbb{R}^{n\times n}$ be symmetric with the ordered eigenvalues $\lambda_i, i = 1, \ldots, n$, and $\mu \in \mathbb{R}$. Function

$$\nu : \mathbb{N} \to \mathbb{N}, \quad \nu(\mu) := \#\{i \,|\, \lambda_i < \mu, \lambda_i \in \sigma(M)\},$$

computes the number of eigenvalues of M in the interval $(-\infty, \mu)$. This function allows to compute a particular eigenvalue of M, by "slicing the spectrum". Suppose that the eigenvalue λ_l is of interest and that function $\nu(\cdot)$ is known. Then λ_l can be computed by the following bisection method:

i) Find a starting interval $[a, b]$, such that $\nu(a) < l \leq \nu(b)$.

ii) Compute the evaluation in the middle point $\nu(\frac{a+b}{2})$. If $\nu(\frac{a+b}{2}) > l$, proceed with the computation in the interval $[a, \frac{a+b}{2}]$, otherwise in $[\frac{a+b}{2}, b]$.

iii) Repeat step ii), until reaching a sufficiently small interval.

The authors propose to evaluate $\nu(\cdot)$ by computing an approximate LDL decomposition

of $M - \mu I$. In particular, for a given M, the LDL decomposition [60]

$$M - \mu I = LDL^T,$$

with L lower triangular and D diagonal, is a congruence transformation. The number of negative entries in D corresponds to $\nu(\mu)$, because of Sylvester's inertia law [60]. The implementation of one LDL decomposition in the HODLR format requires $\mathcal{O}(k^2 n \log^2 n)$ operations and consequently, the computational cost for obtaining one eigenvalue scales as $\mathcal{O}(k^2 n \log^4 n)$. The authors also address the computation of the method in the HSS format. However, this method does not compute eigenvectors.

For general hierarchical matrices, the bisection method based on approximate LDL decompositions is possible. However, it might occur that the diagonal matrix D is not congruent to $M - \mu I$, as a result of truncation errors, as shown in [18]. Additionally, this method has been extended to the case of $\mathcal{H}^2$-matrices; see [16]. The computational cost per eigenvalue in that case is $\mathcal{O}(n \log n)$.

3.5 Divide-and-conquer methods

The main idea of the methods presented in this section is based on the classical divide-and-conquer method for symmetric tridiagonal matrices [36]. This method recursively divides a symmetric tridiagonal matrix T into a block diagonal matrix plus a rank-1 update as follows:

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

with $T_1 \in \mathbb{R}^{n_1 \times n_1}$ and $T_2 \in \mathbb{R}^{n_2 \times n_2}$ symmetric tridiagonal and $b \in \mathbb{R}^n$. Suppose that spectral decompositions $T_1 = Q_1 \Lambda_1 Q_1^T$ and $T_2 = Q_2 \Lambda_2 Q_2^T$ are obtained recursively. Then T can be written as

$$T = \operatorname{diag}(Q_1, Q_2) \Big(\operatorname{diag}(\Lambda_1, \Lambda_2) \ + \ \alpha \tilde{b} \tilde{b}^T \Big) \operatorname{diag}(Q_1, Q_2)^T,$$

with $\tilde{b} = \operatorname{diag}(Q_1^T, Q_2^T) b$. The computation of the spectral decomposition of T is completed once the eigenvalues and eigenvectors of the matrix $\operatorname{diag}(\Lambda_1, \Lambda_2) + \alpha \tilde{b} \tilde{b}^T$ are obtained. Employing a stable and efficient method proposed in [64] for the computation of eigenvectors, the divide-and-conquer method exhibits $\mathcal{O}(n^2)$ complexity for computing the

complete spectral decomposition.

3.5.1 Divide-and-conquer for $\mathcal{H}_{n\times n}(1)$

We now describe a divide-and-conquer method proposed in [61] for HODLR matrices with the off-diagonal rank one.

Consider a symmetric $M \in \mathcal{H}_{n\times n}(1)$ of level p, given as

$$M = \begin{bmatrix} M_1 & uv^T \\ vu^T & M_2 \end{bmatrix},$$

with $M_1, M_2 \in \mathcal{H}_{\frac{n}{2}\times\frac{n}{2}}(1)$ of level $p-1$ and $u, v \in \mathbb{R}^{\frac{n}{2}}$. If the eigenvalue problems for $M_1 = Q_1\Lambda_1 Q_1^T$ and $M_2 = Q_2\Lambda_2 Q_2^T$ are computed by recursion, then updating M by a similarity transformation $\mathrm{diag}(Q_1, Q_2)$ yields

$$\tilde{M} = \begin{bmatrix} \Lambda_1 & \tilde{u}\tilde{v}^T \\ \tilde{v}\tilde{u}^T & \Lambda_2 \end{bmatrix}, \text{ with } \quad \tilde{u} = Q_1 u, \tilde{v} = Q_2 v.$$

Unlike for the standard tridiagonal eigenvalue problem, the authors in [61] propose a rank-2 update to a block diagonal matrix:

$$\tilde{M} = \mathrm{diag}(Q_1, Q_2)\Big(\mathrm{diag}(\Lambda_1, \Lambda_2) + aa^T - bb^T\Big)\,\mathrm{diag}(Q_1, Q_2)^T.$$

The eigenvalues are computed using bisection and Newton's method, while the eigenvectors can be computed by the inverse iteration. The complexity of the presented algorithm is $\mathcal{O}(n^2)$ for computing the spectrum.

3.5.2 Divide-and-conquer for HSS matrices

Recently, a new method based on the classical divide-and-conquer approach has been proposed in [111] for the computation of a complete spectral decomposition. In particular, the authors in [111] extend the classical divide-and-conquer approach to HSS matrices. Unlike for the method described in the previous section, here the authors employ a strategy which involves a rank-k update to a block diagonal matrix, instead of a more straightforward rank-$2k$ update. More specifically, for a symmetric HSS matrix $M \in \mathbb{R}^{n\times n}$

with off-diagonal rank k, the divide step is performed as follows

$$M = \mathrm{diag}(\tilde{M}_1, \tilde{M}_2) + BB^T, \tag{3.1}$$

where $\tilde{M}_1, \tilde{M}_2$ are HSS matrices of size $n/2$ and level $p-1$, and $B \in \mathbb{R}^{n\times k}$. Moreover, $\tilde{M}_1$ and $\tilde{M}_2$ are rank-k updates to the starting diagonal blocks. The conquer step is performed analogously to the classical divide-and-conquer method, but by applying k times a rank-1 update as a result of (3.1).

The proposed procedure exploits the nestedness of bases in the HSS format, as well as the fast-multipole method to speed up the computations. The computational complexity of the method is $\mathcal{O}(k^2 n \log n) + \mathcal{O}(kn \log^2 n)$. Additionally, the matrix of eigenvectors is returned in factored form, requiring $\mathcal{O}(kn \log n)$ memory.

4 Fast computation of spectral projectors of banded matrices

Given a symmetric banded matrix $A \in \mathbb{R}^{n\times n}$ with eigenvalues

$$\lambda_1 \leq \cdots \leq \lambda_\nu < \mu < \lambda_{\nu+1} \leq \cdots \leq \lambda_n,$$

we consider the computation of the spectral projector $\Pi_{<\mu}(A)$ associated with the eigenvalues $\lambda_1, \ldots, \lambda_\nu$. We specifically target the situation where both n and ν are large, say $n = 100\,000$ and $\nu = 50\,000$, which makes approaches based on computing eigenvectors computationally expensive. For a tridiagonal matrix, the MRRR algorithm requires $\mathcal{O}(\nu n)$ operations and memory [44] to compute the ν eigenvectors needed to define $\Pi_{<\mu}(A)$.

There are a number of applications giving rise to the problem under consideration. First and foremost, this task is at the heart of linear scaling methods for the calculation of the electronic structure of molecules with a large number of atoms. For insulators at zero temperature, the density matrix is the spectral projector associated with the eigenvalues of the Hamiltonian below the so called HOMO-LUMO gap[1]; see [59] for an overview. The Hamiltonian is usually symmetric and, depending on the discretization and the structure of the molecule, it can be (approximately) banded. A number of existing linear scaling methods use that this sometimes implies that the spectral projector may also admit a good approximation by a banded matrix; see [21] for a recent survey and a mathematical justification. For this approach to work well, the HOMO-LUMO gap should not become too small. For metallic systems, this gap converges to zero, which makes it impossible to apply an approach based on approximate bandedness or, more generally, sparsity.

[1] HOMO = highest occupied molecular orbital; LUMO = lowest unoccupied molecular orbital

Another potential important application for banded matrices arises in dense symmetric eigenvalue solvers. The eigenvalues and eigenvectors of a symmetric dense matrix A are usually computed by first reducing A to tridiagonal form and then applying the QR algorithm, divide-and-conquer method or MRRR; see, e.g. [6, 42] for recent examples. It is by no means trivial to implement the reduction to tridiagonal form efficiently so that it performs well on a modern computing architecture with a memory hierarchy. Most existing approaches [5, 26, 71, 73, 105], with the notable exception of [96], are based on successive band reduction [27]. In this context, it would be preferable to design an eigenvalue solver that works directly with banded matrices, bypassing the need for tridiagonal reduction. While we are not aware of any such extension of MRRR, this possibility has been explored several times for the divide-and-conquer method, e.g., in [4, 72]. The variants proposed so far seem to suffer either from numerical instabilities or from a complexity that grows significantly with the bandwidth. The method proposed in this work can be used to directly compute the spectral projector of a banded matrix, which in turn can be used as a basis for a fast spectral divide-and-conquer algorithm in the spirit of Nakatsukasa and Higham [91].

To deal with matrices exhibiting a small relative gap, defined as

$$\mathsf{gap} = (\lambda_{\nu+1} - \lambda_\nu)/(\lambda_n - \lambda_1),$$

one needs to go beyond sparsity. It turns out that hierarchical matrices are much better suited in such a setting. Intuitively, this can be well explained by considering the approximation of the Heaviside function $\Pi_{<\mu}(x)$ on the eigenvalues of A. While a polynomial approximation of $\Pi_{<\mu}$ corresponds to a sparse approximation of $\Pi_{<\mu}(A)$ [21], a rational approximation corresponds to an approximation of $\Pi_{<\mu}(A)$ that features hierarchical low-rank structure. It is well known, see, e.g., [95], that a rational approximation is more powerful in dealing with nearby singularities, such as $x = \mu$ for $\Pi_{<\mu}(x)$.

In addition to the existing approaches that use hierarchical low-rank structures for the fast computation of matrix functions discussed in Chapter 2, we mention that Beylkin, Coult, and Mohlenkamp [25] proposed a combination of the Newton–Schulz iteration with the HODLR format to compute spectral projectors for banded matrices. However, the algorithm does not fully exploit the potential of low-rank formats, as it converts a full matrix to the HODLR format in each iteration of the algorithm.

Other hierarchical matrix techniques for eigenvalue problems include slicing-the-spectrum, which uses LDL^T decompositions to compute eigenvalues in a specified interval for symmetric HODLR and HSS matrices [18] as well as $\mathcal{H}^2$–matrices [16]. Approximate $\mathcal{H}$–matrix inverses can be used as preconditioners in iterative eigenvalue solvers; see [83, 85] for examples. Recently, Vogel et al. [111] developed a fast divide-and-conquer method for computing all eigenvalues and eigenvectors in the HSS format. However, as the matrix of eigenvectors is represented in a factored form, it would be a nontrivial and possibly expensive detour to compute spectral projectors via this approach. We refer to Chapter 3 for an overview.

We propose a new method based on a variant [91] of the QR-based dynamically weighted Halley algorithm (QDWH) for computing a polar decomposition [89]. Our method exploits the fact that the iterates of QDWH applied to a banded matrix can be well approximated in the HODLR format. In fact, we show that the memory needed for storing the approximate spectral projector depends only logarithmically on the spectral gap, a major improvement over approximate sparsity. We discuss in detail the implementation of QDWH in the HODLR format, and in particular, we focus on the efficient and accurate representation of the first iterate, which One major contribution of this chapter is to show how this can be done efficiently.

The remainder of the chapter is organized as follows. In Section 4.1, we review the QDWH algorithm for computing a spectral projector $\Pi_{<\mu}(A)$. In Section 4.2 we derive new a priori bounds on the singular values for off-diagonal blocks of $\Pi_{<\mu}(A)$ based on the best rational approximation to the sign function, from which we deduce bounds on the memory required to store $\Pi_{<\mu}(A)$ approximately in the HODLR format. Section 4.3 discusses the efficient realization of the QR decomposition required in the first iterate of the QDWH algorithm. Section 4.4 summarizes our newly proposed QDWH algorithm in the HODLR format and provides implementation details. Finally, numerical experiments both for tridiagonal and banded matrices are shown in Section 4.5.

The content presented in this chapter is based on the published article [81].

4.1 Computation of spectral projectors via QDWH

In the following, we assume $\mu = 0$ without loss of generality, and thus consider the computation of the spectral projector $\Pi_{<0}(A)$ associated with the negative eigenvalues of a symmetric nonsingular matrix $A \in \mathbb{R}^{n\times n}$. We first start by recalling several essential definitions.

Let $A = Q\Lambda Q^T$ be a spectral decomposition of A such that $\Lambda = \operatorname{diag}(\Lambda_-, \Lambda_+)$, where Λ_- and Λ_+ are diagonal matrices containing the ν negative and the $n-\nu$ positive eigenvalues of A, respectively. The spectral projector associated with the negative eigenvalues is given by

$$\Pi_{<0} = Q \begin{bmatrix} I_\nu & 0 \\ 0 & 0 \end{bmatrix} Q^T, \tag{4.1}$$

while the spectral projector related to the positive eigenvalues is defined by

$$\Pi_{>0} = Q \begin{bmatrix} 0 & 0 \\ 0 & I_{n-\nu} \end{bmatrix} Q^T. \tag{4.2}$$

Additionally, the closely related matrix sign function is defined by

$$\operatorname{sign}(A) = Q \begin{bmatrix} -I_\nu & 0 \\ 0 & I_{n-\nu} \end{bmatrix} Q^T. \tag{4.3}$$

As the matrix sign function is in the core of our computations, in the following theorem we present a summary of some important properties of $\operatorname{sign}(A)$, and for completeness provide its proof.

Theorem 4.1. (Theorem 5.1. in [77]) *Let $A \in \mathbb{R}^{n\times n}$ symmetric matrix with no zero eigenvalues and let $S = \operatorname{sign}(A)$. Then*

i) $S^2 = I$ (S is involutory);

ii) S is diagonalizable with eigenvalues ± 1;

iii) A and S commute, i.e. $SA = AS$;

iv) $\frac{1}{2}(I - S)$ and $\frac{1}{2}(I + S)$ are spectral projectors onto the invariant subspaces associated with the negative and positive eigenvalues, respectively.

Proof. i) The involutory property follows from the definition (4.3) of S and the orthogonality of matrix Q:

$$S^2 = Q \begin{bmatrix} -I_\nu & 0 \\ 0 & I_{n-\nu} \end{bmatrix} Q^T Q \begin{bmatrix} -I_\nu & 0 \\ 0 & I_{n-\nu} \end{bmatrix} Q^T = QQ^T = I.$$

ii) From the definition (4.3) follows that the eigenvalues of S are ± 1 and the matrix of eigenvectors is Q.

iii) Writing A in terms of its spectral decomposition $A = QDQ^T$ leads to

$$AS = QD \begin{bmatrix} -I_\nu & 0 \\ 0 & I_{n-\nu} \end{bmatrix} Q^T = Q \begin{bmatrix} -I_\nu & 0 \\ 0 & I_{n-\nu} \end{bmatrix} DQ^T = SA,$$

where we utilize the property that two diagonal matrices commute. We also note the commutation property holds for general matrix functions.

iv) We first show that $\frac{1}{2}(I - S)$ corresponds to $\Pi_{<0}(A)$ given in (4.1). Inserting the definition of S yields

$$\frac{1}{2}(I - S) = \frac{1}{2} Q \left(I - \begin{bmatrix} -I_\nu & 0 \\ 0 & I_{n-\nu} \end{bmatrix} \right) Q^T = Q \begin{bmatrix} I_\nu & 0 \\ 0 & 0 \end{bmatrix} Q^T = \Pi_{<0}(A),$$

where the last equality holds because of (4.1). Analogously, we obtain that $\frac{1}{2}(I + S)$ equals to $\Pi_{>0}(A)$ given in (4.2).

□

Following [91], our approach for computing $\Pi_{<0}(A)$ is based on a well-known connection to the polar decomposition. The polar decomposition [60, Chapter 9] of A takes the form $A = UH$ for an orthogonal matrix U and a symmetric positive definite matrix H. Given the spectral decomposition of A

$$\begin{aligned} A &= Q\Lambda Q^T = Q \operatorname{diag}(\Lambda_-, \Lambda_+) Q^T \\ &= \underbrace{Q \operatorname{diag}(-I_\nu, I_{n-\nu}) Q^T}_{=:U} \cdot \underbrace{Q \operatorname{diag}(|\Lambda_-|, |\Lambda_+|) Q^T}_{=:H} \end{aligned}$$

the polar decomposition of A follows. In particular, this shows that the matrix sign

function sign(A) coincides with the orthogonal factor U from the polar decomposition. More importantly, $\Pi_{<0}(A) = \frac{1}{2}(I - U)$. Similarly, the spectral projector associated with the positive eigenvalues equals to $\Pi_{>0}(A) = \frac{1}{2}(I + U)$.

4.1.1 QDWH algorithm

In this section we recall the QDWH algorithm for computing the polar decomposition of A proposed in [89], likewise its variant proposed in [91]. In particular, we mostly follow the exposition given in [89].

Starting from the Halley iteration for computing the orthogonal polar factor U of A

$$\begin{aligned} X_0 &= A, \\ X_{k+1} &= X_k(3I + X_k^T X_k)(I + 3X_k^T X_k)^{-1}, \quad k \geq 0, \end{aligned} \tag{4.4}$$

in order to enhance its convergence, the authors in [89] consider the *dynamically weighted Halley* (DWH) iteration:

$$\begin{aligned} X_0 &= A/\alpha, \\ X_{k+1} &= X_k(a_k I + b_k X_k^T X_k)(I + c_k X_k^T X_k)^{-1}, \quad k \geq 0, \end{aligned} \tag{4.5}$$

with $\alpha = \|A\|_2$ and nonnegative scalars a_k, b_k, c_k. These weighting parameters are chosen such that the value l_{k+1}, depending on a_k, b_k, c_k, is maximized and satisfies that the interval $[l_{k+1},\, 1]$ contains all singular values of the iterate X_{k+1}.

To show how the weighting parameters need to be chosen, let us first examine the behavior of the singular values of the iterates X_k. To this end, suppose that $X_k = W\Sigma_k Z^T$ is the singular value decomposition of X_k, and let l_k satisfy

$$[\sigma_{\min}(X_k),\, \sigma_{\max}(X_k)] \subseteq [l_k,\, 1] \subset [0,\, 1]\,, \tag{4.6}$$

given that the smallest singular value of the initial iterate satisfies $\sigma_{\min}(X_0) = 1/\kappa(A) \equiv l_0$. After applying one step of the DWH iteration (4.5), we can express X_{k+1} as

$$X_{k+1} = W \underbrace{\Sigma_k(a_k I + b_k \Sigma_k^2)(I + c_k \Sigma_k^2)^{-1}}_{\Sigma_{k+1}} Z^T. \tag{4.7}$$

Therefore the singular values of X_{k+1} are given in terms of the singular values of X_k, with

$$\sigma_i(X_{k+1}) = r_k(\sigma_i(X_k)), \quad i = 1, \cdots, n, \tag{4.8}$$

where r_k is a rational function of type $(3, 2)$ given as

$$r_k(x) = x\frac{a_k + b_k x^2}{1 + c_k x^2}. \tag{4.9}$$

In particular, (4.7) and (4.8) show that the iterates in (4.5) have mutual singular vectors, while the singular values are defined via mappings r_k, as a diagonal matrix Σ_{k+1} satisfies

$$\Sigma_{k+1} = r_k(\Sigma_k) = \cdots = r_k(\cdots(r_1(\Sigma_0))\cdots),$$

with the matrix function on the right-hand side defined in the classical sense.

From (4.13) and (4.8) it follows that

$$[\sigma_{\min}(X_{k+1}),\, \sigma_{\max}(X_{k+1})] \subseteq \left[\min_{x\in[l_k,1]} r_k(x),\, \max_{x\in[l_k,1]} r_k(x)\right]. \tag{4.10}$$

Since the polar factor U is an orthogonal matrix, all its singular values equal to 1, and it is therefore a reasonable way to examine the distance of the iterate X_{k+1} from U by measuring how far are the iterate's singular values from 1. Hence, a choice of parameters a_k, b_k, c_k ought to be such that two following conditions are satisfied:

$$i) \quad 0 < r_k(x) \leq 1, \quad \text{for } x \in [l_k,\, 1], \tag{4.11}$$

$$ii) \max_{a_k,b_k,c_k} \min_{x\in[l_k,1]} r_k(x). \tag{4.12}$$

Finally, when parameters a_k, b_k, c_k satisfying (4.11) and (4.12) are computed, for the singular values of the iterate X_{k+1} we obtain

$$[\sigma_{\min}(X_{k+1}),\, \sigma_{\max}(X_{k+1})] \subseteq [l_{k+1},\, 1] \subset [0,\, 1]\,, \quad \text{with } l_{k+1} = \min_{x\in[l_k,1]} r_k(x). \tag{4.13}$$

The computation of the parameters a_k, b_k, c_k requires solving the optimization prob-

lem (4.11) and (4.12). The solution of this optimization problem is given by

$$a_k = h(l_k), \quad b_k = (a_k - 1)^2/4, \quad c_k = a_k + b_k - 1, \tag{4.14}$$

where the function h is defined as

$$h(l) = \sqrt{1+\gamma} + \frac{1}{2}\sqrt{8 - 4\gamma + \frac{8(2-l^2)}{l^2\sqrt{1+\gamma}}}, \quad \gamma = \sqrt[3]{\frac{4(1-l^2)}{l^4}}.$$

The parameter l_k is determined by the recurrence

$$l_k = l_{k-1}(a_{k-1} + b_{k-1}l_{k-1}^2)/(1 + c_{k-1}l_{k-1}^2), \quad k \geq 1, \quad l_0 = 1/\kappa(A). \tag{4.15}$$

Moreover, it can be shown that when l_k converges to 1, the weighting parameters converge to $(a_k, b_k, c_k) \to (3, 1, 3)$, which are the weighting parameters of the Halley iteration (4.4).

Remark 4.2. Although in this presentation we use the exact singular values of A when deriving the parameters of the DWH iteration, in practice this is often not doable. Instead of using $\alpha = \|A\|_2$ and $l_0 = 1/\kappa(A)$, it is sufficient to obtain their estimates $\tilde{\alpha}$ and $\tilde{l}_0$, respectively, with the property

$$[\sigma_{\min}(A/\tilde{\alpha}),\, 1] \subseteq \left[\tilde{l}_0, 1\right] \subset [0,\, 1].$$

The efficient estimation of the parameters α and l_0, required to start the recurrence (4.5), will be discussed in Section 4.4.

The following result reveals the asymptotic behavior of the DWH iteration.

Theorem 4.3. (Theorem 3.1. in [89]) *For a nonsingular matrix A, the iterates X_k generated by the DWH iteration* (4.5) *converge to the polar factor U of A. Moreover, the asymptotic convergence of the iteration is cubic.*

To derive a meaningful stopping criterion for the DWH iteration, and the overall number of iterations required for convergence, we measure the distance of the iterates from the polar factor U

$$\|X_k - U\|_2 = |1 - \sigma_{\min}(X_k)| = |1 - l_k|.$$

The smallest integer k with the property $|1 - l_k| < u^{-1}$, where $u \approx 1.1 \cdot 10^{-16}$ is the unit roundoff in IEEE double-precision arithmetics, gives an upper bound on the number of

iterations needed for the recurrence (4.5) to converge to the polar factor U of A. This number can be deduced from the scalar recursion (4.15), using a starting parameter $l_0 = 1/\kappa(A)$. Table 4.1 summarizes the number of iterations with respect to a matrix condition number.

Table 4.1 – The number of iterations k of the recurrence (4.5) *with respect to the condition number of a starting matrix ($\kappa(A)$) [89].*

$\kappa(A)$	10^1	10^2	10^5	10^8	10^{10}	10^{12}	10^{16}
k	3	4	5	5	5	5	6

Results in Table 4.1 suggest that the DWH iteration converges within at most six steps for any matrix with the condition number $\kappa(A) \leq 10^{-16}$.

QR-based iteration

The recurrence (4.5) has the equivalent form

$$X_0 = A/\alpha, \tag{4.16a}$$

$$X_{k+1} = \frac{b_k}{c_k} X_k + \frac{1}{\sqrt{c_k}} \left(a_k - \frac{b_k}{c_k} \right) Q_1 Q_2^T, \tag{4.16b}$$

with the QR decomposition

$$\begin{bmatrix} \sqrt{c_k} X_k \\ I \end{bmatrix} = \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} R. \tag{4.17}$$

Throughout this work, we refer to the iteration (4.16) as a *QR-based iteration*. For a symmetric matrix A, the arithmetic cost of one QR-based iteration amounts to $4.5n^3$ flops, by taking into account the structure of matrices in (4.17); we refer the reader to [91] for a more detailed analysis.

The DWH iteration implemented in terms of (4.16) is referred to as the *QDWH algorithm*. For completeness, we provide the pseudocode in Algorithm 4.1.

Algorithm 4.1 QDWH algorithm

Input: Nonsingular matrix $A \in \mathbb{R}^{n\times n}$.
Output: Approximation $U \in \mathbb{R}^{n\times n}$ to the orthogonal polar factor of A.

1: Set the initial parameters α, l_0.
2: $X_0 = A/\alpha$.
3: $k = 0$.
4: **repeat**
5: Compute a_k, b_k, c_k according to the recurrence (4.14).
6: Compute the QR decomposition $\begin{bmatrix}\sqrt{c_k}X_k \\ I\end{bmatrix} = \begin{bmatrix}Q_1 \\ Q_2\end{bmatrix} R$.
7: Compute $X_{k+1} = \frac{b_k}{c_k}X_k + \left(a_k - \frac{b_k}{c_k}\right) Q_1 Q_2^T$.
8: $k = k + 1$.
9: Compute $l_k = l_{k-1}(a_{k-1} + b_{k-1}l_{k-1}^2)/(1 + c_{k-1}l_{k-1}^2)$.
10: **until** convergence
11: $U = X_k$.

Cholesky-based iteration

As observed in [91], the DWH recurrence (4.5) can also be rewritten in an equivalent way involving the Cholesky decomposition. In particular, the *Cholesky-based iteration* of the DWH recurrence is given by

$$Z_k = I + c_k X_k^T X_k, \;\; W_k = \mathrm{chol}(Z_k), \tag{4.18a}$$

$$X_{k+1} = \frac{b_k}{c_k}X_k + \left(a_k - \frac{b_k}{c_k}\right)(X_k W_k^{-1})W_k^{-T}, \tag{4.18b}$$

where $\mathrm{chol}(Z_k)$ denotes the Cholesky factor of Z_k. For a symmetric matrix A, the implementation of one Cholesky-based iteration (4.18) requires in total $3n^3 - n^3/6$ flops. This includes the computation of a symmetric positive definite matrix Z_k requiring $n^3/2$ operations, the computation of its Cholesky factor W_k with $n^3/3$ operations, and finally solving two triangular linear systems with $2n^3$ operations. Thus, this variant of the DWH recurrence requires less arithmetic operations than the evaluation of the QR-based iteration.

Additionally, we mention that a higher–order variant of QDWH, called Zolo-pd, has recently been proposed by Freund and Nakatsukasa [90]. This method approximates the polar decomposition in at most two iterations but requires more arithmetic operations per iteration.

4.1.2 Switching between QR-based and Cholesky-based iterations

Because of its lower operation count, it can be expected that one Cholesky-based iteration (4.18) is faster than one QR-based iteration (4.16). However, when Z_k is ill-conditioned, which is signaled by a large value of c_k, the numerical stability of (4.18) can be compromised. To avoid this, it is proposed in [91] to switch from QR-based iteration (4.16) to Cholesky-based iteration (4.18) as soon as $c_k \leq 100$. Since c_k converges monotonically from above to 3, this implies that this hybrid approach will first perform a few QR-based iterations and then switch for good to Cholesky-based iterations. In fact, numerical experiments presented in [91] indicate that at most two QR-based iterations are performed.

For reasons explained in Section 4.2 below, we prefer to perform only one QR-based iteration and then switch to Cholesky-based iterations. To explore the impact of this choice on numerical accuracy, we perform a comparison of the QDWH algorithm proposed in [91] with a variant of QDWH that performs only one QR-based iteration. We consider the following error measures:

$$\begin{aligned} e_{\mathrm{id}}^{Q} &:= \|U^2 - I\|_2, \\ e_{\mathrm{trace}}^{Q} &:= |\operatorname{trace}(U) - \operatorname{trace}(\operatorname{sign}(A))|, \\ e_{\mathrm{SP}}^{Q} &:= \left\| \frac{1}{2}(I - U) - \Pi_{<0}(A) \right\|_2, \end{aligned} \tag{4.19}$$

where U denotes the output of the QDWH algorithm, and $\Pi_{<0}(A)$ the spectral projector returned by the MATLAB function `eig`.

Example 4.4. Let $A \in \mathbb{R}^{2000\times 2000}$ be a symmetric tridiagonal matrix constructed as described in Section 4.5.1, such that half of the spectrum of A is contained in $[-1, -\mathsf{gap}]$ and the other half in $[\mathsf{gap}, 1]$, for $\mathsf{gap} \in \{10^{-1}, 10^{-5}, 10^{-10}, 10^{-15}\}$. As can be seen in Table 4.2, the errors obtained by both variants of the QDWH algorithm exhibit a similar behavior. Even for tiny values of gap, no significant loss of accuracy is observed if only one QR-based iteration is performed.

We would like to emphasize that Example 4.4 provides numerical evidence but does *not* prove that one QR iteration is enough to obtain a numerically stable algorithm. As the error analysis from [91] does not allow to draw this conclusion, it remains open to perform an analysis that supports our choice.

Table 4.2 – Comparison of errors in the QDWH algorithm with one or several QR-based iterations.

Algorithm [91]	gap	10^{-1}	10^{-5}	10^{-10}	10^{-15}
one QR-based iteration (4.16)	e^Q_{trace}	$5.55 \cdot 10^{-17}$	$7.22 \cdot 10^{-16}$	$2.22 \cdot 10^{-16}$	$1.11 \cdot 10^{-16}$
	e^Q_{id}	$1.15 \cdot 10^{-15}$	$2.41 \cdot 10^{-15}$	$1.84 \cdot 10^{-15}$	$1.82 \cdot 10^{-15}$
	e^Q_{SP}	$1.87 \cdot 10^{-14}$	$4.35 \cdot 10^{-12}$	$1.88 \cdot 10^{-6}$	$1.91 \cdot 10^{-2}$
several QR-based iterations (4.16)	e^Q_{trace}	$5.55 \cdot 10^{-17}$	$1.22 \cdot 10^{-15}$	$1.53 \cdot 10^{-16}$	$6.25 \cdot 10^{-16}$
	e^Q_{id}	$1.15 \cdot 10^{-15}$	$2.58 \cdot 10^{-15}$	$1.81 \cdot 10^{-15}$	$2.04 \cdot 10^{-15}$
	e^Q_{SP}	$1.87 \cdot 10^{-14}$	$2.12 \cdot 10^{-12}$	$2.82 \cdot 10^{-6}$	$3.06 \cdot 10^{-2}$
	# of (4.16)	1	2	2	3

4.2 Hierarchical matrix approximation of spectral projectors

In Section 2.1.2 we have discussed the storage of spectral projectors of banded matrices within the HODLR and a more general $\mathcal{H}$–matrix format, generated by the admissibility condition motivated by the 1D integral equations. In particular, the analysis provided in Example 2.2 has shown that it is more beneficial to use the HODLR format for the computation of spectral projectors of banded matrices than the hierarchical format presented in Section 2.1.2.

To implement a variant of the QDWH algorithm for computing an approximation of a spectral projector in the HODLR format, we use algorithms from Section 2.2. The computation of a Cholesky-based iteration (4.18) can be completely efficiently carried out within the HODLR format. However, the QR-based iteration (4.16) of QDWH requires the computation of the QR decomposition (4.17). As discussed in Section 2.2.8, unlike for hierarchical Cholesky decomposition, there is no straightforward way of performing QR decompositions in hierarchical matrix arithmetics. Hence, instead of using any of the existing algorithms, we develop a novel method in Section 4.3 to compute the QR decomposition (4.16) that exploits the particular structure of the matrix in the first iteration of the QDWH algorithm. This allows us to represent the matrices Q_1 and Q_2 from (4.17) exactly in the HODLR format.

In the following we derive a theoretical storage complexity for spectral projectors given in

the HODLR format and show its logarithmic dependence on the spectral gap.

4.2.1 A priori bounds on singular values and memory requirements

To study the approximation of $\Pi_{<0}(A)$ in the HODLR format, we first derive bounds for the singular values of the off-diagonal ranks based on rational approximations to the sign function. In the following, we say that a rational function r is of type (k, s) and write $r \in \mathcal{R}_{k,s}$ if $r = p/q$ holds for polynomials p and q of degree at most k and s, respectively.

Rational approximation of sign function

Given $0 < c < d < \infty$, the min-max problem

$$\min_{r \in \mathcal{R}_{m,m}} \max_{x \in [-d,-c] \cup [c,d]} |\operatorname{sign}(x) - r(x)|, \quad \operatorname{sign}(x) = \begin{cases} 1, & x \in [c,\, d], \\ -1, & x \in [-d,\, -c], \end{cases} \tag{4.20}$$

has a unique solution s_m [1, Chapter 9]), the so-called Zolotarev function, which takes the form

$$s_m(x) := Mx \frac{\prod_{i=1}^{\lfloor (m-1)/2 \rfloor} (x^2 + c_{2i})}{\prod_{i=1}^{\lfloor m/2 \rfloor} (x^2 + c_{2i-1})}.$$

The coefficients c_i are given in terms of the Jacobi elliptic function $\operatorname{sn}(\cdot; \kappa)$:

$$c_i = c^2 \frac{\operatorname{sn}^2(\frac{iK(\kappa)}{m}; \kappa)}{1 - \operatorname{sn}^2(\frac{iK(\kappa)}{m}; \kappa)}, \tag{4.21}$$

where $\kappa = \sqrt{1 - (c/d)^2}$ and K is defined as the complete elliptic integral of the first kind

$$K(x) = \int_0^{\frac{\pi}{2}} \frac{d\theta}{\sqrt{1 - x^2 \sin^2 \theta}} = \int_0^1 \frac{dt}{\sqrt{(1-t^2)(1-x^2t^2)}}, \quad 0 \le x \le 1.$$

The constant M is uniquely determined by the condition

$$\min_{x \in [-d,-c]} 1 + s_m(x) = \max_{x \in [c,d]} 1 - s_m(x).$$

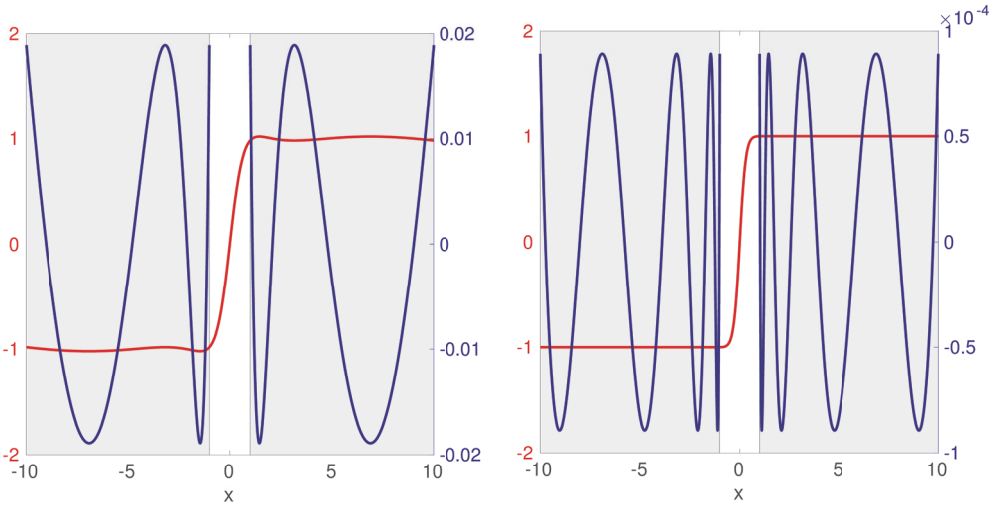

Figure 4.1 – Zolotarev's rational approximations for the domain $[-10,-1]\cup[1,10]$. *The lines colored red are the best rational* $\mathcal{R}_{m,m}$ *approximation to the sign function on* $[-10,10]$, *while blue lines represent the approximation errors on* $[-10,-1]$ *and* $[-1,10]$. *Left: The best rational approximation in* $\mathcal{R}_{4,4}$. *Right: The best rational approximation in* $\mathcal{R}_{8,8}$. *These plots were produced by using RKToolbox [24].*

Let us define by E_m the maximal approximation error on the domain $[-d,-c]\cup[c,d]$:

$$E_m := \max_{x\in[-d,-c]\cup[c,d]} |\operatorname{sign}(x) - s_m(x)|.$$

As shown in [15], lower and upper bounds on E_m are given by

$$\frac{4\rho^{-m}}{(1+\rho^{-2m})^4} \leq E_m \leq 4\rho^{-m}, \tag{4.22}$$

where $\rho = \rho(\kappa) = \exp\left(\frac{\pi K(\sqrt{1-\kappa^2})}{K(\kappa)}\right)$.

The following lemma derives a bound from (4.22) that reveals the influence of the gap on the approximation error E_m.

Lemma 4.5. *With the notation introduced above and* $\mathsf{gap} = c/d$, *it holds that*

$$E_m \leq 4\exp\left(-\frac{\pi^2 m}{2\log\left(4/\mathsf{gap}+2\right)}\right). \tag{4.23}$$

Proof. Following Braess and Hackbusch [31], we have

$$K\left(\sqrt{1-\kappa^2}\right) = K(\mathsf{gap}) \geq \pi/2,$$
$$K(\kappa) = K\left(\sqrt{1-\mathsf{gap}^2}\right) \leq \log(4/\mathsf{gap}+2).$$

Thus, the upper bound in (4.22) implies

$$E_m \leq 4\exp\left(-\frac{\pi^2 m}{2\log(4/\mathsf{gap}+2)}\right).$$

□

It is simple to bound the ranks of the off-diagonal blocks for a rational function applied to a banded matrix. We first recall an important result.

Corollary 4.6. (Corollary 1.36 in [109]) *Suppose $A \in \mathbb{R}^{n\times n}$ is a nonsingular matrix and α, β nonempty subsets of $N := \{1, 2, \ldots, n\}$, with $|\alpha| < n$ and $|\beta| < n$. Then*

$$\operatorname{rank}(A^{-1}(\alpha,\beta)) = \operatorname{rank}(A(N\backslash\beta, N\backslash\alpha)) + |\alpha| + |\beta| - n,$$

where $A(\alpha,\beta)$ denotes a submatrix of A with row indices α and columns indices β. In particular,

$$\operatorname{rank}(A^{-1}(\alpha, N\backslash\alpha)) = \operatorname{rank}(A(\alpha, N\backslash\alpha)).$$

This corollary states that rank of the off-diagonal blocks is preserved under inversion.

Lemma 4.7. *Consider a b-banded matrix $A \in \mathbb{R}^{n\times n}$ and a rational function r_m of type (m,m), with poles disjoint from the spectrum of A. Then the off-diagonal blocks of $r_m(A)$ have rank at most mb.*

Proof. Assuming that r_m has simple poles, let $r_m(x) = \sum_{i=1}^{m} \omega_i (x-\mu_i)^{-1}$ be a partial fraction expansion of r_m, with $\omega_i, \mu_i \subset \mathbb{C}, i = 1, \ldots, m$. Thus, $r_m(A)$ is a sum of m shifted inverses of A. By Corollary 4.6, the off-diagonal blocks of each summand $B = A - \mu_i I$ satisfy

$$\operatorname{rank} B^{-1}|_{\text{off}} = \operatorname{rank} B|_{\text{off}}.$$

Noting that $\operatorname{rank} B|_{\text{off}} = b$, because B has bandwidth b, this completes the proof for the case of simple poles.

Now, suppose that r_m has non-simple poles. Because the poles of r_m are disjoint from the eigenvalues of A by assumption, there is for every $\varepsilon > 0$ a rational function $\tilde{r}_m$ of type (m, m) with simple poles (still disjoint from the spectrum of A) such that $\|r_m(A) - \tilde{r}_m(A)\|_2 \leq \varepsilon$. Since every off-diagonal block of $\tilde{r}_m(A)$ has rank bounded by mb, it follows from the semi-continuity of the rank function that every off-diagonal block of $r_m(A)$ has rank at most mb by letting $\varepsilon \to 0$. □

It is worth noting that, since the coefficients c_{2i-1} are all distinct, a partial fraction expansion of a Zolotarev function s_m always has simple poles. Moreover, it is interesting to remark that Nakatsukasa and Freund [90] noted that the rational function (4.9) in the QDWH algorithm is, up to a scaling factor, the best rational approximation of type $(3, 2)$ to the sign function on $[-1, -l_0] \cup [l_0, 1]$. Moreover, since a composition of Zolotarev functions is a Zolotarev function [90], the mapping function in the kth iteration of the QDWH algorithm is the best rational approximation to the sign function of type $(3^k, 3^k - 1)$ on $[-1, -l_0] \cup [l_0, 1]$, which provides a theoretical justification for the fast convergence of the QDWH algorithm.

Singular value decay of off-diagonal blocks

The results of Lemma 4.5 and Lemma 4.7 allow us to establish exponential decay for the singular values of the off-diagonal blocks in $\Pi_{<0}(A)$ or, equivalently, in $\operatorname{sign}(A)$ for any symmetric banded matrix A. By rescaling A, we may assume without loss of generality that its spectrum is contained in $[-d, -c] \cup [c, d]$. We let $\sigma_i(\cdot)$ denote the ith largest singular value of a matrix.

Theorem 4.8. *Consider a symmetric b-banded matrix $A \in \mathbb{R}^{n \times n}$ with the eigenvalues contained in $[-d, -c] \cup [c, d]$, and $m \in \mathbb{N}$. Letting $\mathsf{gap} = c/d$, the singular values of any off-diagonal block $\Pi_{<0}(A)|_{\text{off}}$ satisfy*

$$\sigma_{mb+1}(\Pi_{<0}(A)|_{\text{off}}) \leq 2 \exp\left(- \frac{\pi^2 m}{2 \log(4/\mathsf{gap} + 2)} \right). \tag{4.24}$$

Proof. Let s_m denote the solution of the min-max problem (4.20). Because $s_m(A)|_{\text{off}}$ has rank at most mb by Lemma 4.7, and the best rank-i approximation error is governed by

the $(i+1)$th largest singular value, it follows from (4.23) that

$$\sigma_{mb+1}(\operatorname{sign}(A)|_{\text{off}}) \leq \|\operatorname{sign}(A) - s_m(A)\|_2 \leq \max_{x\in[-d,-c]\cup[c,d]} |\operatorname{sign}(x) - s_m(x)|$$
$$\leq 4\exp\left(-\frac{\pi^2 m}{2\log(4/\mathsf{gap}+2)}\right).$$

The statement therefore follows from the relation $\Pi_{<0}(A)|_{\text{off}} = -\frac{1}{2}\operatorname{sign}(A)|_{\text{off}}$. ∎

Example 4.9. This example gives an insight on the quality of the bound obtained in Theorem 4.8. For this purpose, we construct, as explained in Section 4.5.1, symmetric b-banded matrices of size $n = 2000$ with eigenvalues uniformly distributed in $[-1, -\mathsf{gap}]\cup[\mathsf{gap}, 1]$, with half of the spectrum negative, and the other half positive. In particular, we consider two matrices: a tridiagonal matrix with $\mathsf{gap} = 10^{-1}$, and a 4-banded matrix with $\mathsf{gap} = 10^{-4}$. In both cases the spectral projector $\Pi_{<0}$ is computed using MATLAB's built-in function `eig`. The decay in the singular values of the upper off-diagonal blocks $\Pi_{<0}(1:n/2, n/2+1:n)$ is reported in Figure 4.2, together with the theoretical bound (4.24). In case of a large relative spectral gap, the bound is tight and nicely follows the decay of singular values; see Figure 4.2 (left). However, for a smaller spectral gap, in Figure 4.2 (right) we notice that a rate of the decay in the theoretical bound is slower than the singular value decay.

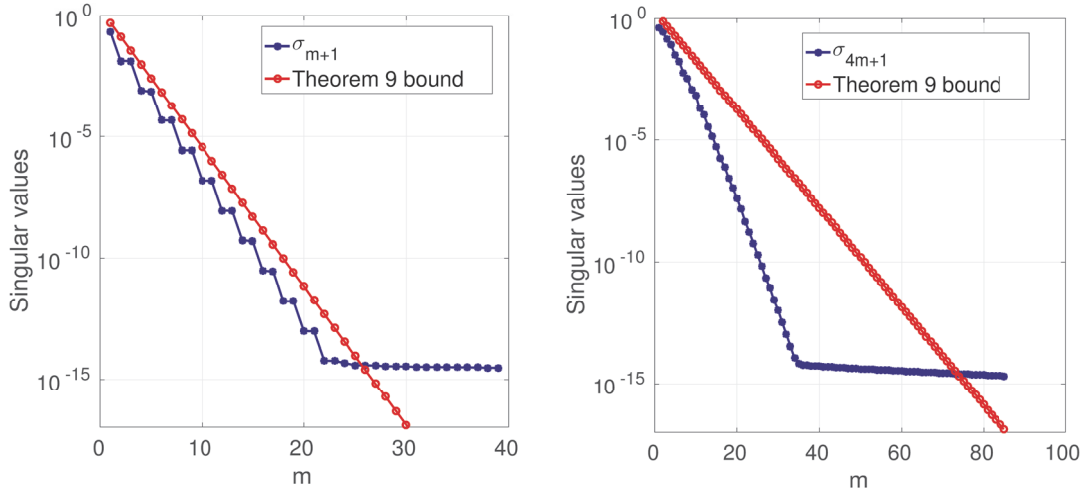

Figure 4.2 – Example 4.9. Comparison of the singular value decay in the off-diagonal blocks of a spectral projector $\Pi_{<0}(A)$ with the theoretical bound from Theorem 4.8. Left: A is a tridiagonal matrix with a spectral gap 10^{-1}. Right: A is 4-banded matrix with a spectral gap 10^{-4}.

Memory requirements with respect to gap

Theorem 4.8 allows us to study the memory required to approximate $\Pi_{<0}(A)$ in the HODLR format to a prescribed accuracy. For this purpose, let $\Pi^{\mathcal{H}}$ denote the best approximation in the Frobenius norm of $\Pi_{<0}(A)$ in the HODLR format with all off-diagonal ranks bounded by mb. Necessarily, the diagonal blocks of $\Pi^{\mathcal{H}}$ and $\Pi_{<0}(A)$ are the same. For an off-diagonal block of size k, Theorem 4.8 implies

$$\begin{aligned}\|\Pi_{<0}(A)|_{\text{off}} - \Pi^{\mathcal{H}}|_{\text{off}}\|_F^2 = \sum_{i=mb+1}^{k} \sigma_i(\Pi_{<0}(A)|_{\text{off}})^2 &\leq \sum_{j=m}^{\lceil k/b\rceil - m} b\,\sigma_{jb+1}(\Pi_{<0}(A)|_{\text{off}})^2 \\ &\leq 4b \sum_{j=m}^{\lceil k/b\rceil - m} \tau^{2j} \leq \frac{4b}{1-\tau^2}\tau^{2m},\end{aligned}$$

with $\tau = \exp\left(-\frac{\pi^2}{2\log(4/\mathsf{gap}+2)}\right)$. Taking into account the total number of off-diagonal blocks, we arrive at

$$\|\Pi_{<0}(A) - \Pi^{\mathcal{H}}\|_F^2 \leq \frac{8b}{1-\tau^2}(n/n_{\min} - 1)\tau^{2m}.$$

Thus, the value of m needed to attain $\|\Pi_{<0}(A) - \Pi^{\mathcal{H}}\|_F \leq \delta$ for a desired accuracy $\delta > 0$ satisfies $m = \mathcal{O}\Big(|\log \mathsf{gap}| \cdot \log\Big(bn\delta^{-1}|\log \mathsf{gap}|\Big)\Big)$. The corresponding approximation $\Pi^{\mathcal{H}}$ requires

$$\mathcal{O}\Big(|\log \mathsf{gap}| \cdot \log\Big(bn\delta^{-1}|\log \mathsf{gap}|\Big) bn \log n\Big) \tag{4.25}$$

memory. Up to a double logarithmic factor, this shows that the memory depends logarithmically on the spectral gap.

Comparison to approximate sparsity

We now compare (4.25) with known results for approximate sparsity. Assuming we are in the setting of Theorem 4.8, it is shown in [21] that the off-diagonal entries of $\Pi_{<0}(A)$ satisfy

$$|(\Pi_{<0}(A))_{ij}| \leq C\,\mathrm{e}^{-\alpha|i-j|}, \qquad \alpha = \frac{1}{2b}\log\left(\frac{1+\mathsf{gap}}{1-\mathsf{gap}}\right),$$

for some constant $C > 0$ depending on c and d. Let $\Pi^{(m)}$ denote the best approximation in the Frobenius norm to $\Pi_{<0}(A)$ by a matrix of bandwidth m. Following [21, Theorem 7.7], we obtain

$$\|\Pi_{<0}(A) - \Pi^{(m)}\|_F \leq \frac{C}{\sqrt{\alpha}} \sqrt{n}\, \mathrm{e}^{-\alpha m}\,.$$

Choosing a value of m that satisfies $m = \mathcal{O}\Big(b\,\mathsf{gap}^{-1} \log\Big(Cbn\delta^{-1}\,\mathsf{gap}^{-1}\Big)\Big)$ thus ensures an accuracy of $\delta > 0$, where we used $\alpha \approx \mathsf{gap}\,/b$. As the storage of $\Pi^{(m)}$ requires $\mathcal{O}(mn)$ memory, we arrive at

$$\mathcal{O}\left(\frac{1}{\mathsf{gap}} \log\Big(Cbn\delta^{-1}\,\mathsf{gap}^{-1}\Big) bn\right) \tag{4.26}$$

memory. In contrast to the logarithmic dependence in (4.25), the spectral gap now enters the asymptotic complexity inverse proportionally. On the other hand, (4.25) features a factor $\log n$ that is not present in (4.26). For most situations of practical interest, we expect that the much milder dependence on the gap far outweighs this additional factor. Hence, the comparison between (4.25) and (4.26) provides strong theoretical justification for favoring the HODLR format over approximate sparsity.

4.3 QR-based first iteration of QDWH

The first QR-based iteration of the QDWH algorithm requires computing the QR decomposition

$$\begin{bmatrix} cA \\ I \end{bmatrix} = \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} R \tag{4.27}$$

for some scalar $c > 0$. Without loss of generality, we suppose that $c = 1$. In this section, we develop an algorithm that requires $\mathcal{O}(b^2 n)$ operations for performing this decomposition when A is a b–banded matrix and storing Q_1 and Q_2 implicitly. This is then used to directly compute Q_1 and Q_2 in the HODLR format with $\mathcal{O}(bn \log n)$ operations. Since it is significantly simpler, we first discuss the case of a tridiagonal matrix A before treating the case of general b.

It is interesting to note that the need for computing a QR decomposition of the form (4.27)

also arises in the solution of ill-posed inverse problems with Tikhonov regularization; see, e.g., [28]. However, when solving ill-posed problems, usually only the computation of the upper triangular factor R is required, while the QDWH algorithm requires the computation of the orthogonal factor.

4.3.1 QR decomposition of $\begin{bmatrix} A \\ I \end{bmatrix}$ for tridiagonal A

For the case of a bidiagonal matrix A, Eldén [49] proposed a fast algorithm for reducing a matrix $\begin{bmatrix} A \\ I \end{bmatrix}$ to upper triangular form. In the following, we propose a modification of Eldén's algorithm suitable for tridiagonal A.

Our proposed algorithm is probably best understood from the illustration in Figure 4.3 for $n = 4$. In the ith step of the algorithm, all subdiagonal elements in the ith column of $\begin{bmatrix} A \\ I \end{bmatrix}$ are annihilated by performing Givens rotations either with the diagonal element, or with the element $(n+1, i)$. By carefully choosing the order of annihilation, only one new nonzero subdiagonal element is created in column $i+1$. The detailed pseudocode of this procedure is provided in Algorithm 4.2. We use $G(i, j, \alpha)$ to denote a Givens rotation of angle α that is applied to rows/columns i and j.

Algorithm 4.2 Fast QR decomposition (4.27) for tridiagonal A

Input: Tridiagonal matrix A.
Output: Factors Q, R of a QR decomposition of $\begin{bmatrix} A \\ I \end{bmatrix}$.

$Q \leftarrow I_{2n}, R \leftarrow \begin{bmatrix} A \\ I \end{bmatrix}$.
Construct $G(1, n+1, \beta_1)$ to annihilate $R(n+1, 1)$.
Update $R \leftarrow G(1, n+1, \beta_1)^T R$ and $Q \leftarrow QG(1, n+1, \beta_1)$
Construct $G(1, 2, \gamma_1)$ to annihilate $R(2, 1)$.
Update $R \leftarrow G(1, 2, \gamma_1)^T R$ and $Q \leftarrow QG(1, 2, \gamma_1)$.
for $i = 2, \ldots, n$ **do**
    Construct $G(n+1, n+i, \alpha_i)$ to annihilate $R(n+i, i)$.
    Update $R \leftarrow G(n+1, n+i, \alpha_i)^T R$ and $Q \leftarrow QG(n+1, n+i, \alpha_i)$.
    Construct $G(i, n+1, \beta_i)$ to annihilate $R(n+1, i)$.
    Update $R \leftarrow G(i, n+1, \beta_i)^T R$ and $Q \leftarrow QG(i, n+1, \beta_i)$.
    if $i < n$ **then**
        Construct $G(i, i+1, \gamma_i)$ to annihilate $R(i+1, i)$.
        Update $R \leftarrow G(i, i+1, \gamma_i)^T R$ and $Q \leftarrow QG(i, i+1, \gamma_i)$.
    end if
end for

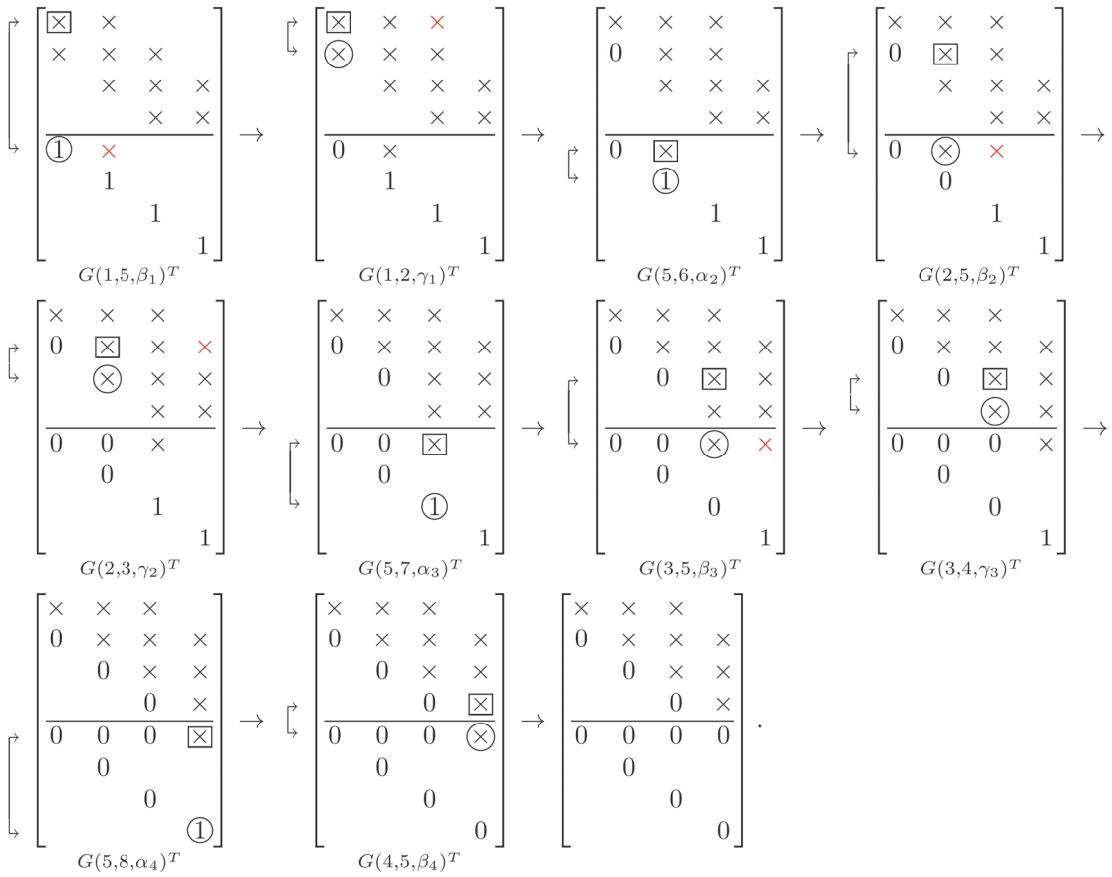

Figure 4.3 – Fast QR decomposition of $\begin{bmatrix} A \\ I \end{bmatrix}$ *for tridiagonal* A *and* $n = 4$*. In each step, a Givens rotation is applied to the rows denoted by the arrows. Crosses denote generically nonzero elements, boxed/circled crosses are used to define Givens rotations, while red crosses denote the fill-in during the current operation.*

Algorithm 4.2 performs $3n - 2$ Givens rotations in total: two in the first and in the last column, and three in each of the remaining $n - 2$ columns. By exploiting its sparsity in a straightforward manner, only $\mathcal{O}(n)$ operations and memory are required to compute the upper triangular factor R. The situation is more complicated for the orthogonal factor. Since Q is dense, it would require $\mathcal{O}(n^2)$ operations and memory to form Q using Algorithm 4.2. In the following section, we explain how the low-rank structure of Q can be exploited to reduce this cost to $\mathcal{O}(n \log n)$.

Ranks of off-diagonal blocks and fast computation of orthogonal factor

For our purposes, it suffices to compute the first n columns of the $2n \times 2n$ matrix Q, that is, the $n \times n$ matrices $Q_1 = Q(1:n, 1:n)$ and $Q_2 = Q(n+1:2n, 1:n)$. The order of Givens rotations in Algorithm 4.2 implies that Q_1 is an upper Hessenberg matrix, while Q_2 is an upper triangular matrix. The result of Theorem 4.10 implies that all off-diagonal blocks of Q_1 and Q_2 have rank at most two.

Theorem 4.10. *For the orthogonal factor Q returned by Algorithm 4.2, it holds that the matrices $Q(1:k, k+1:n)$ and $Q(n+1:n+k, k+1:n)$ have rank at most two for all $1 \leq k < n$.*

Proof. We only prove the result for $Q(1:k, k+1:n)$; the proof for $Q(n+1:n+k, k+1:n)$ is analogous.

During steps $1, \ldots, k-1$ of Algorithm 4.2, $Q(1:k, k+1:n)$ is not modified and remains zero. In step k of Algorithm 4.2, column $k+1$ of Q is modified, while $Q(1:k, k+2:n)$ remains zero. After step k has been completed, let us set

$$\mathcal{U} := \operatorname{span}\{Q(1:k, k+1), Q(1:k, n+1)\} \subset \mathbb{R}^k. \tag{4.28}$$

By construction, $\operatorname{span} Q(1:k, k+1:n) \subset \mathcal{U}$. In the following, we show by induction that this relation holds for all subsequent steps of Algorithm 4.2. Suppose that $\operatorname{span} Q(1:k, k+1:n) \subset \mathcal{U}$ holds after i steps for some i with $k \leq i \leq n-1$. In step $i+1$, the following is performed:

1. $G(n+1, n+i+1, \alpha_{i+1})$ is applied to columns $n+1$ and $n+i+1$ of Q. Because $Q(1:k, n+i+1)$ is zero before applying the rotation, this simply effects a rescaling of column $n+1$ and thus $Q(1:k, n+1) \in \mathcal{U}$ remains true.
2. $G(i+1, n+1, \beta_{i+1})$ is applied to columns $i+1$ and $n+1$ of Q, which preserves $\operatorname{span} Q(1:k, k+1:n) \subset \mathcal{U}$.
3. If $i < n$, $G(i+1, i+2, \gamma_{i+1})$ is applied to columns $i+1$ and $i+2$ of Q, which again preserves $\operatorname{span} Q(1:k, k+1:n) \subset \mathcal{U}$.

After completion of the algorithm, the column span of $Q(1:k, k+1:n)$ is thus contained in a subspace of dimension at most two. This proves the statement of the theorem. □

The QDWH algorithm makes use of the matrix product $Q_1 Q_2^T$, see (4.16b). Theorem 4.10, together with the upper Hessenberg and upper triangular structure, directly implies that the ranks of the lower and upper off-diagonal blocks of $Q_1 Q_2^T$ are bounded by three and two, respectively. In fact, the following theorem shows a slightly stronger result.

Theorem 4.11. *Every off-diagonal block of $Q_1 Q_2^T$ for the orthogonal factor returned by Algorithm 4.2 has rank at most 2.*

Proof. By Algorithm 4.2 and Theorem 4.10, the matrices Q_1 and Q_2 admit for any $1 \leq k < n$ a partitioning of the from

$$Q_1 = \left[\begin{array}{c|c} X_1 & U_1 V_1^T \\ \hline \sigma e_1 e_k^T & X_2 \end{array}\right], \quad Q_2 = \left[\begin{array}{c|c} Y_1 & U_2 V_2^T \\ \hline 0 & Y_2 \end{array}\right],$$

where $X_1 \in \mathbb{R}^{k \times k}, X_2 \in \mathbb{R}^{n-k \times n-k}$ are upper Hessenberg, $Y_1 \in \mathbb{R}^{k \times k}, Y_2 \in \mathbb{R}^{n-k \times n-k}$ are upper triangular, $U_1, U_2 \in \mathbb{R}^{k \times 2}$, $V_1, V_2 \in \mathbb{R}^{n-k \times 2}$, $\sigma \in \mathbb{R}$, and e_1, e_k denote unit vectors of appropriate lengths. The upper off-diagonal block of $Q_1 Q_2^T$ equals to a product of rank-2 matrices

$$(Q_1 Q_2^T)(1:k, k+1:n) = X_1 \cdot 0 + U_1 \underbrace{V_1^T Y_2^T}_{\widetilde{V}_1^T} = U_1 \widetilde{V}_1^T.$$

Moreover, the lower off-diagonal block amounts to a sum of a rank-1 and a rank-2 matrix

$$\begin{aligned}(Q_1 Q_2^T)(k+1:n, 1:k) &= \sigma e_1 e_k^T Y_1^T + \underbrace{X_2 V_2}_{\widetilde{V}_2} U_2^T \\ &= \sigma e_1 Y_1(:,k)^T + \widetilde{V}_2 U_2^T.\end{aligned}$$

If $\sigma = 0$, the statement holds. Otherwise, we first show that the vectors $Y_1(:,k)$ and $U_2(:,1)$ are collinear. Let us recall that the vectors $Y_1(:,k)$, $U_2(:,1)$ coincide with the vectors $Q(n+1:n+k+1,k)$, $Q(n+1:n+k+1,k+1)$ computed during step k of Algorithm 4.2. As $Q(n+1:n+k+1,k)$ and $Q(n+1:n+k+1,k+1)$ are collinear after performing step k, the same holds for $Y_1(:,k)$, $U_2(:,1)$, and $Y_1(:,k) = \eta U_2(:,1)$ for some $\eta \in \mathbb{R}$. Hence, we obtain

$$(Q_1 Q_2^T)(k+1:n, 1:k) = \sigma\eta e_1 U_2(:,1)^T + \widetilde{V}_2 U_2^T = \widehat{V}_2 U_2^T,$$

which completes the proof. □

We note that Theorem 4.11 and the recurrence (4.16) imply that the first iterate of the QDWH algorithm can be exactly represented in the HODLR format with off-diagonal ranks at most 3.

Fast computation of factors Q_1 and Q_2

The fast computation of factors Q_1 and Q_2 needed in the QR-based iteration (4.16) begins after all Givens rotations in Algorithm 4.2 had been generated, and stored as a sequence of 2×2 matrices $G(\alpha)$, where

$$G(\alpha) = \begin{pmatrix} \cos(\alpha) & \sin(\alpha) \\ -\sin(\alpha) & \cos(\alpha) \end{pmatrix}.$$

We start by setting $Q_1 = Q(1:n, 1:n) = I_n$ and $Q_2 = Q(n+1:2n, 1:n) = 0_n$ in the HODLR format. The computation of the diagonal blocks, which are stored as dense matrices, of the HODLR matrices Q_1 and Q_2 is obtained by applying Givens rotations to the columns of the corresponding block from the right. Following Algorithm 4.2, in each step which affects a diagonal block, at most two columns of the block are modified and column $Q(:, n+1)$ is updated. This implies the computation of $p \times p$ diagonal blocks in Q_1 and Q_2 requires only $\mathcal{O}(p^2)$ arithmetic operations.

On the other hand, the proof of Theorem 4.10 can be turned into a procedure for directly computing low-rank representations for the off-diagonal blocks of Q_1 and Q_2 in the HODLR format. As a result the structure of Q_1 and Q_2, all lower off-diagonal blocks have ranks 1 and 0 respectively, and the computation of their low-rank representations is straightforward. We therefore only consider the computation of low-rank representations for the upper off-diagonal blocks in Q_1 and Q_2. The detailed pseudocode is given in Algorithm 4.3, and an illustration of the algorithm is provided in Figure 4.4.

The upper off-diagonal $p \times s$ blocks in Q_1 and Q_2 correspond to $Q(r+1:r+p, k+1:k+s)$, $k+s \leq n$, for $r+p \leq k$ and $r-n+p \leq k$, respectively. The construction of their rank–2 representations UV^T begins in step k of Algorithm 4.2. During step k only vector $Q(r+1:r+p, k+1)$ is modified and becomes a scalar multiple of $Q(r+1:r+p, k)$. After step k is completed, following (4.28), we set $U = [Q(r+1:r+p, k+1), Q(r+1:r+p, n+1)]$.

As V stores the coefficients of the columns of the block in the basis U, in steps $k+1,\ldots,k+s$ of Algorithm 4.2 we apply Givens rotations only to the rows of V. Moreover, knowing that $Q(r+1:r+p,n+1)\in\text{span}\{U(:,1),U(:,2)\}$ in steps $k+1,\ldots,k+s$ an additional vector is needed to keep its coefficients in the basis U updated. Hence, using Algorithm 4.3, the overall complexity for computing a low-rank representation of an upper-diagonal $p\times s$ block is $\mathcal{O}(\max\{p,s\})$.

Algorithm 4.3 Computation of an off-diagonal block $Q(r+1:r+p,k+1:k+s)$

Input: Givens rotations generated by Algorithm 4.2, vectors $Q(r+1:r+p,k+1)$, $Q(r+1:r+p,n+1)$ obtained after computing $Q(r+1:r+p,k)$ in Algorithm 4.2.

Output: Rank–2 representation UV^T, with $U\in\mathbb{R}^{p\times 2}, V\in\mathbb{R}^{s\times 2}$, of the off-diagonal block, updated vectors $Q(r+1:r+p,k+s+1)$ and $Q(r+1:r+p,n+1)$.

$U\leftarrow[Q(r+1:r+p,k+1),Q(r+1:r+p,n+1)]$.
$V\leftarrow[e_1,0]\in\mathbb{R}^{s\times 2}$.
$v_{n+1}\leftarrow[0,1]\in\mathbb{R}^2$.
for $j=1,\ldots,s$ **do**
  Update $V(s+1:,)\leftarrow\cos(\alpha_{j+k})V(s+1,:)$.
  Update $\begin{bmatrix}V(j,:)\\ v_{n+1}\end{bmatrix}\leftarrow G(\beta_{j+k})^T\begin{bmatrix}V(j,:)\\ v_{n+1}\end{bmatrix}$.
  if $j+k<n$ **then**
    if $j<s$ **then**
      Update $V([j,j+1],:)\leftarrow G(\gamma_{j+k})^T V([j,j+1],:)$.
    else
      Update $Q(r+1:r+p,k+s+1)\leftarrow\sin(\gamma_{s+k})\Big(U(:,1)V(s,1)+U(:,2)V(s,2)\Big)$.
      Update $V(s,:)\leftarrow\cos(\gamma_{s+k})V(s,:)$.
    end if
  end if
end for
Update $Q(r+1:r+p,n+1)\leftarrow U(:,1)v_{n+1}(1)+U(:,2)v_{n+1}(2)$.
Return U and V.
Return $Q(r+1:r+p,n+1)$.

The overall computational complexity for computing Q_1 in the HODLR format is derived as follows. For simplicity we assume that $n=2^p n_{\min}$. Then the diagonal blocks are computed with $\mathcal{O}(n)$ operations. Moreover, on each level of HODLR subdivision, there are 2^{l-1} upper off-diagonal blocks, with low-rank factors of size $n/2^l\times 2$. By Algorithm 4.3, each off-diagonal block can be computed with linear complexity. Summing over all upper

off-diagonal blocks, the number of performed operations amounts to

$$\sum_{l=1}^{p} 2^{l-1}\mathcal{O}\left(\frac{n}{2^l}\right) = p\mathcal{O}(n) \approx \mathcal{O}(n \log n).$$

Therefore, the number of operations needed to compute Q_1 in the HODLR format is $\mathcal{O}(n \log n)$. The computational analysis for Q_2 is derived in an analogous way.

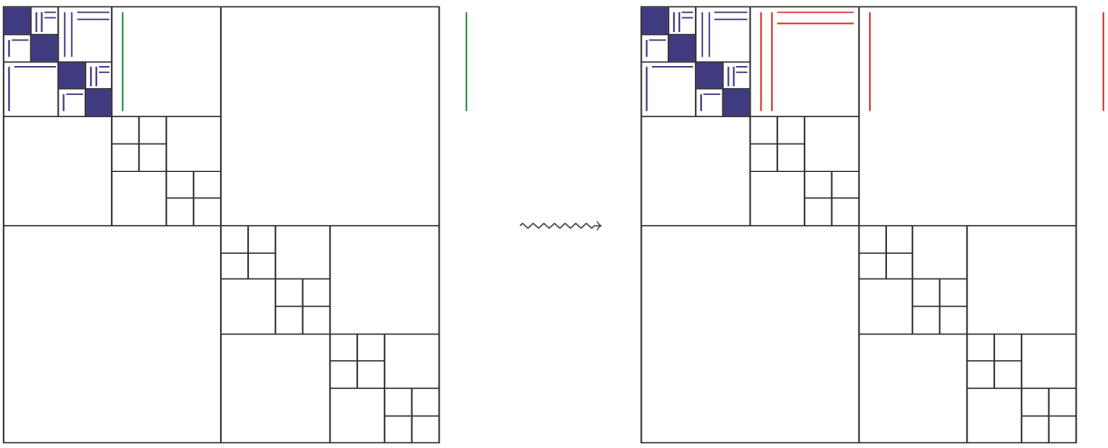

Figure 4.4 – The computation of one upper off-diagonal block in Q_1 using Algorithm 4.3. Left: Q_1 prior to the execution of Algorithm 4.3. Right: Q_1 after the execution of Algorithm 4.3. Blocks denoted blue are computed before the execution of Algorithm 4.3, and remain unchanged while performing Algorithm 4.3. Columns denoted green are the input of Algorithm 4.3, while columns and rows denoted red are the output of Algorithm 4.3.

4.3.2 QR decomposition of $\begin{bmatrix} A \\ I \end{bmatrix}$ for banded A

In this section, we discuss the QR decomposition of $\begin{bmatrix} A \\ I \end{bmatrix}$ for a banded symmetric matrix A with bandwidth $b > 1$. Let us first note that Eldén [50] proposed a fast algorithm for reducing a matrix $\begin{bmatrix} A \\ L \end{bmatrix}$ with an upper triangular banded matrix L. Eldén's algorithm does not cover the fast computation of the orthogonal factor and requires the application of $(2b+1)n + nb - \frac{5}{2}b^2 - \frac{3}{2}b$ Givens rotations. In the following, we propose a different algorithm that requires a fewer number of Givens rotations. In particular, the newly proposed algorithm requires $(2b+1)n - b^2 - b$ Givens rotations.

Figure 4.11 illustrates the idea of our algorithm for $n = 6$ and $b = 3$. In the ith step of the algorithm, the subdiagonal elements in the ith column of $\begin{bmatrix} A \\ I \end{bmatrix}$ are annihilated as follows. A first group of Givens rotations $(\alpha_{i,j})$ annihilates all elements in row $n + i$,

which consists of the diagonal element of I and the fill-in introduced in the previous step. Then a Givens rotation (β_i) annihilates the element $(n+1, i)$. Finally, a second group of Givens rotations $(\gamma_{i,j})$ annihilates all subdiagonal elements in the ith column of A. The detailed procedure is given in Algorithm 4.4.

Algorithm 4.4 Fast QR decomposition (4.27) for banded A

Input: Banded matrix A with bandwidth b.
Output: Factors Q, R of a QR decomposition of $\begin{bmatrix} A \\ I \end{bmatrix}$.

1: $Q \leftarrow I_{2n}$, $R \leftarrow \begin{bmatrix} A \\ I \end{bmatrix}$.
2: Construct $G(1, n+1, \beta_1)$ to annihilate $R(n+1, 1)$.
3: Update $R \leftarrow G(1, n+1, \beta_1)^T R$ and $Q \leftarrow QG(1, n+1, \beta_1)$
4: **for** $j = 2, \ldots, b+1$ **do**
5: Construct $G(1, j, \gamma_{1,j})$ to annihilate $R(j, 1)$.
6: Update $R \leftarrow G(1, j, \gamma_{1,j})^T R$ and $Q \leftarrow QG(1, j, \gamma_{1,j})$.
7: **end for**
8: **for** $i = 2, \ldots, n$ **do**
9: Construct $G(n+1, n+i, \alpha_{i,i})$ to annihilate $R(n+i, i)$.
10: Update $R \leftarrow G(n+1, n+i, \alpha_{i,i})^T R$ and $Q \leftarrow QG(n+1, n+i, \alpha_{i,i})$.
11: **for** $j = i+1, \ldots, \min\{n, b+i-1\}$ **do**
12: Construct $G(n+i, n+j, \alpha_{i,j})$ to annihilate $R(n+i, j)$.
13: Update $R \leftarrow G(n+i, n+j, \alpha_{i,j})^T R$ and $Q \leftarrow QG(n+i, n+j, \alpha_{i,j})$.
14: **end for**
15: Construct $G(i, n+1, \beta_i)$ to annihilate $R(n+1, i)$.
16: Update $R \leftarrow G(i, n+1, \beta_i)^T R$ and $Q \leftarrow QG(i, n+1, \beta_i)$.
17: **if** $i < n$ **then**
18: **for** $j = i+1, \ldots, \min\{n, b+i\}$ **do**
19: Construct $G(i, j, \gamma_{i,j})$ to annihilate $R(j, i)$.
20: Update $R \leftarrow G(i, j, \gamma_{i,j})^T R$ and $Q \leftarrow QG(i, j, \gamma_{i,j})$.
21: **end for**
22: **end if**
23: **end for**

Ranks of off-diagonal blocks and fast computation of orthogonal factor

Due to the order of annihilation in Algorithm 4.4, it follows that $Q_1 = Q(1:n, 1:n)$ is a b-Hessenberg matrix (that is, the matrix is zero below the bth subdiagonal) while $Q_2 = Q(n+1:2n, 1:n)$ is an upper triangular matrix. The following result and its proof yield an $\mathcal{O}(b^2 n) + \mathcal{O}(bn \log n)$ algorithm for computing Q_1 and Q_2 in the HODLR format, analogous to Theorem 4.10, Section 4.3.1 and Algorithm 4.3.

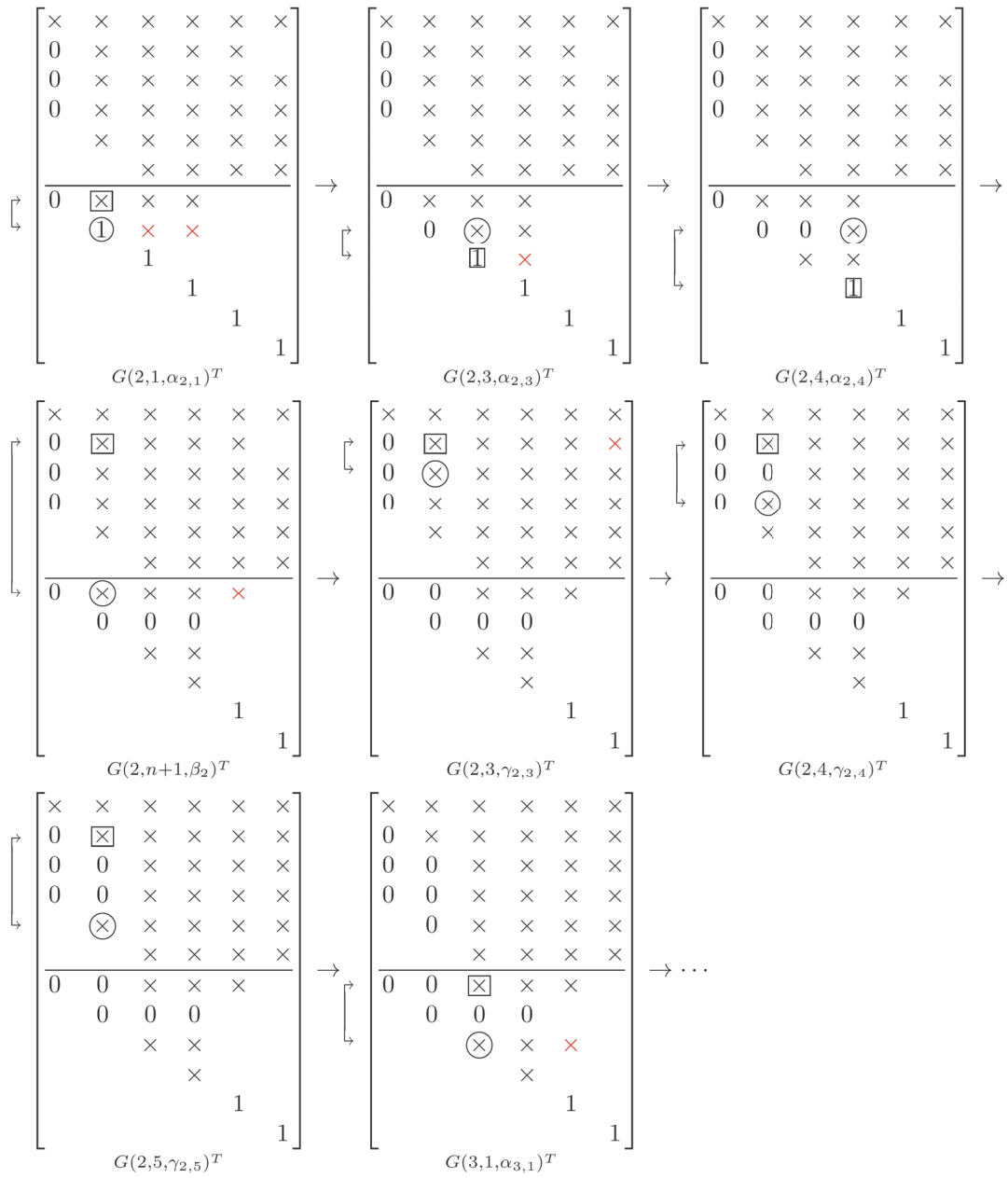

Figure 4.5 – Second step of fast QR decomposition (Algorithm 4.4) of $\begin{bmatrix} A \\ I \end{bmatrix}$ *for banded* A *with* $n = 6$ *and* $b = 3$*. In each step, a Givens rotation is applied to the rows denoted by the arrows. Crosses denote generically nonzero elements, boxed/circled crosses are used to define Givens rotations, while red crosses denote the fill-in during the current operation.*

Theorem 4.12. *For the orthogonal factor Q returned by Algorithm 4.4, it holds that the matrices $Q(1:k, k+1:n)$ and $Q(n+1:n+k, k+1:n)$ have rank at most $2b$ for all $1 \leq k < n$.*

Proof. Again, we prove the result for $Q(1:k, k+1:n)$ only. After k steps of Algorithm 4.4 have been performed, we define the subspace

$$\begin{aligned}\mathcal{U} := \operatorname{span}\{&Q(1:k, k+1), \ldots, Q(1:k, k+b), Q(1:k, n+1),\\ &Q(1:k, n+k+1), \ldots, Q(1:k, n+k+b-1)\},\end{aligned}$$

which is of dimension not larger than $2b$. At this point, the columns $Q(1:k, j)$ are zero for $j = k+b+1, \ldots, n$ and $j = n+k+b, \ldots, 2n$. Thus,

$$\operatorname{span} Q(1:k, k+1:n+1) \subset \mathcal{U}, \qquad \operatorname{span} Q(1:k, n+k+1:2n) \subset \mathcal{U} \tag{4.29}$$

hold after k steps of Algorithm 4.4. We now show by induction that this relation holds for all subsequent steps.

Suppose that (4.29) holds after i steps with $k \leq i \leq n-1$. In step $i+1$, the following operations are performed by Algorithm 4.4:

1. $G(n+1, n+i+1, \alpha_{i+1,i+1})$ is applied to columns $n+1$ and $n+i+1$ of Q, which affects and preserves both inclusions in (4.29). Then $G(n+i+1, n+j, \alpha_{i+1,j})$ is applied to columns $n+i+1$ and $n+j$ of Q, for $j = i+2 : \min\{n, i+b\}$, hence $\operatorname{span} Q(1:k, n+k+1:2n) \subset \mathcal{U}$ remains true.
2. $G(i+1, n+1, \beta_{i+1})$ is applied to columns $i+1$ and $n+1$ of Q, preserving $\operatorname{span} Q(1:k, k+1:n+1) \subset \mathcal{U}$.
3. If $i+1 < n$, $G(i+1, j, \gamma_{i+1,j})$ is applied to columns $i+1$ and j of Q, for $j = i+2 : \min\{n, i+b+1\}$, which retains $\operatorname{span} Q(1:k, k+1:n+1) \subset \mathcal{U}$.

Therefore (4.29) holds after Algorithm 4.4 has been completed, which completes the proof of the theorem. □

The following result is an extension of Theorem 4.13 from the tridiagonal to the banded case.

Theorem 4.13. *Every off-diagonal block of $Q_1 Q_2^T$ for the orthogonal factor returned by Algorithm 4.4 has rank at most $2b$.*

Proof. We first note that for $k < 2b$ the statement readily follows, as one dimension of the off-diagonal blocks is in this case smaller than $2b$; thus the same holds for ranks.

On the other hand, similarly as in Theorem 4.11, for any k such that $1 \leq 2b \leq k < n$ Algorithm 4.4 and Theorem 4.12 imply that the matrices Q_1 and Q_2 can be decomposed as

$$Q_1 = \left[\begin{array}{c|c} X_1 & U_1 V_1^T \\ \hline R_k E_k^T & X_2 \end{array}\right], \quad Q_2 = \left[\begin{array}{c|c} Y_1 & U_2 V_2^T \\ \hline 0 & Y_2 \end{array}\right],$$

where $X_1 \in \mathbb{R}^{k\times k}, X_2 \in \mathbb{R}^{n-k\times n-k}$ are b-upper Hessenberg matrices, $Y_1 \in \mathbb{R}^{k\times k}, Y_2 \in \mathbb{R}^{n-k\times n-k}$ are upper triangular matrices, $U_1, U_2 \in \mathbb{R}^{k\times 2b}$, $V_1, V_2 \in \mathbb{R}^{n-k\times 2b}$, while $R_k \in \mathbb{R}^{n-k\times b}$ with $R_k(1:b, 1:b)$ upper triangular and $E_k \in \mathbb{R}^{n-k\times b}$ with $E_k^T = [0_{b\times n-k-b}\ I_b]$.

The expression for the upper off-diagonal block immediately yields that its rank is bounded by $2b$:

$$(Q_1 Q_2^T)(1:k, k+1:n) = U_1 (V_1^T Y_2^T) = U_1 \tilde{V}_1^T, \quad \text{with } \tilde{V}_1 = Y_2 V_1 \in \mathbb{R}^{n-k\times 2b}.$$

In the following we prove the statement of the theorem for the lower off-diagonal block. To bound the rank of the lower off-diagonal block given as

$$\begin{aligned}(Q_1 Q_2^T)(k+1:n, 1:k) &= R_k E_k^T Y_1^T + (X_2 V_2) U_2^T, \\ &= R_k Y_1(:, k-b+1:k)^T + \widehat{V}_2 U_2^T, \end{aligned} \tag{4.30}$$

we utilize the relation between the column spaces of matrices Y_1 and U_2. After step k of Algorithm 4.4, vectors $Q(n+1:n+k, k-b+1:k) = Y_1(:, k-b+1:k)$ and $Q(n+1:n+k, k+1:k+b) = U_2(:, 1:b)$ span the same vector space. Therefore there exists a matrix $S \in \mathbb{R}^{b\times b}$ such that $Y_1(:, k-b+1:k) = U_2(:, 1:b)S$, which inserted in (4.30) yields

$$(Q_1 Q_2^T)(k+1:n, 1:k) = \left(R_k S^T + \widehat{V}(:, 1:b)\right) U_2(:, 1:b)^T + \widehat{V}(:, b+1:2b) U_2(:, b+1:2b)^T.$$

Therefore the desired result follows. □

4.4 hQDWH algorithm

In this section we provide an overall algorithm for computing spectral projectors of banded matrices in the HODLR format. Algorithm 4.5 summarizes the hQDWH algorithm proposed in this chapter.

Algorithm 4.5 hQDWH algorithm

Input: Symmetric banded matrix A with bandwidth $b \geq 1$, minimal block-size $n_{\min} \geq 2$, truncation tolerance $\epsilon > 0$, stopping tolerance $\delta > 0$.

Output: Approximation P in the HODLR format to spectral projector $\Pi_{<0}(A)$.

1: Choose initial parameters α, l_0 of QDWH according to (4.31).
2: $X_0 = A/\alpha$.
3: $k = 0$.
4: **while** $|1 - l_k| > \delta$ **do**
5: Compute a_k, b_k, c_k according to the recurrence (4.14).
6: **if** $k = 0$ **then**
7: Apply $\begin{cases} \text{Algorithm 4.2,} & \text{for } b = 1 \\ \text{Algorithm 4.4,} & \text{for } b > 1 \end{cases}$ to $\begin{bmatrix} \sqrt{c_0} X_0 \\ I \end{bmatrix}$ and store resulting array G of Givens rotations.
8: Compute Q_1 and Q_2 from G in the HODLR format, generated by $n_{\min}$, using Section 4.3.1 and Algorithm 4.3.
9: $X_1 = \frac{b_0}{c_0} X_0 + \frac{1}{\sqrt{c_0}} \left(a_0 - \frac{b_0}{c_0} \right) Q_1 Q_2^T$.
10: **else**
11: Compute $W_k = \texttt{hchol}(I + c_k X_k^T *_{\mathcal{H}} X_k)$.
12: Solve upper triangular system $Y_k W_k = X_k$ in the HODLR format using a variant of Algorithm 2.5.
13: Solve lower triangular system $V_k W_k^T = Y_k$ in the HODLR format using Algorithm 2.5.
14: Compute $X_{k+1} = \frac{b_k}{c_k} X_k +_{\mathcal{H}} \left(a_k - \frac{b_k}{c_k} \right) V_k$.
15: **end if**
16: $k = k + 1$.
17: $l_k = l_{k-1}(a_{k-1} + b_{k-1} l_{k-1}^2)/(1 + c_{k-1} l_{k-1}^2)$.
18: **end while**
19: Set $U = X_k$.
20: Return $P = \frac{1}{2}(I - U)$.

In the following, we comment on various implementation details of Algorithm 4.5, as well as its computational complexity.

Line 1 As proposed in [91], the parameters $\alpha \gtrsim \|A\|_2$ and $l_0 \lesssim \sigma_{\min}(X_0)$ needed to start

the QDWH algorithm are estimated as

$$\alpha = \texttt{normest}(A), \quad l_0 = \|A/\alpha\|_1/(\sqrt{n} \cdot \texttt{condest}(A/\alpha)), \tag{4.31}$$

where `normest` and `condest` denote the MATLAB functions for estimating the matrix 2–norm using the power method and the 1–norm condition number using [78], respectively. Both functions exploit that A is sparse and require $\mathcal{O}(bn)$ and $\mathcal{O}(b^2n)$ operations, respectively.

Line 7 Following [89], the algorithm stops when l_k is sufficiently close to 1, which is measured by some stopping tolerance δ.

Lines 7– 9 This part of the algorithm deals with the implementation of the first QR-based iterate (4.16). The generation of Givens rotations by Algorithms 4.2 and 4.4 for reducing $\begin{bmatrix} \sqrt{c_0}X_0 \\ I \end{bmatrix}$ to triangular form has been implemented in a C function, making use of the LAPACK routine `DLARTG`. The function is called via a MEX interface and returns an array G containing the cosines and sines of all rotations. This array is then used in step 8 to generate Q_1 and Q_2 in the HODLR format, whose precise form is defined by the input parameter $n_{\min}$. Symmetry of the first iterate X_1 is enforced in the addition in step 9.

Lines 14– 17 The computation of the kth iterate X_k, $k > 1$, involves the Cholesky decomposition, addition, and the solution of triangular linear systems in the HODLR format. Existing techniques for HODLR matrices have been used for this purpose, introduced in Chapter 2, and repeated recompression with the absolute truncation tolerance ϵ is applied. Symmetry of the kth iterate is enforced in the addition in step 14 and step 17.

Remark 4.14. Algorithm 4.5 extends in a straightforward way to the more general hierarchical matrix format from Section 4.2. The only major difference is the need for converting the matrices after Line 9 from the HODLR to the hierarchical matrix format. This extension of Algorithm 4.5 was used in Example 2.2.

Assuming that all ranks in the off-diagonal blocks are bounded by $k \geq b$, we conclude that Algorithm 4.5 requires $\mathcal{O}(kn \log n)$ memory and $\mathcal{O}(k^2 n \log^2 n)$ operations.

4.5 Numerical experiments

In this section, we demonstrate the performance of our preliminary MATLAB implementation of the hQDWH algorithm. All computations were performed in MATLAB version 2014a on an Intel Xeon CPU with 3.07GHz, 4096 KByte of level 2 cache and 192 GByte of RAM, using a single core. The storage requirements are obtained experimentally, using MATLAB built-in functions.

To measure the accuracy of the QDWH algorithm, we use the functions e^Q_{id}, e^Q_{tracc}, e^Q_{SP} defined in (4.19). The error measures $e^{\mathcal{H}}_{\mathrm{id}}$, $e^{\mathcal{H}}_{\mathrm{trace}}$, $e^{\mathcal{H}}_{\mathrm{SP}}$ for the hQDWH algorithm are defined analogously. For the computation of $e^{\mathcal{H}}_{\mathrm{id}}$ and $e^{\mathcal{H}}_{\mathrm{SP}}$ matrices given in the HODLR format are first transformed into dense matrices, followed by the computation of the matrix 2-norm.

In all experiments, we used the tolerance $\delta = 10^{-15}$ for stopping the QDWH/hQDWH algorithms. Unless stated otherwise, the truncation tolerance for recompression in the HODLR format is set to $\epsilon = 10^{-10}$; the minimal block-size is set to $n_{\min} = 250$ for tridiagonal matrices and $n_{\min} = 500$ for banded matrices.

The performance of the algorithm is tested on various types of matrices, including synthetic examples as well as examples from widely used sparse matrix collections. Firstly, we discuss the fast construction of synthetic test matrices used in our experiments.

4.5.1 Construction of synthetic test matrices

Given a bandwidth b and a prescribed set of eigenvalues $\lambda_1, \ldots, \lambda_n$, we construct a symmetric b–banded matrix with given eigenvalues by an orthogonal similarity transformation of $A = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$. For this purpose, we perform the following operation for $i = n, n-1, \ldots, 2$.

First, a Givens rotation $G(i-1, i, \alpha_i)$ is created by annihilating the second component of the vector $\begin{bmatrix} a_{ii} \\ 1 \end{bmatrix}$. The update $A \leftarrow G(i-1, i, \alpha_i)^T A G(i-1, i, \alpha_i)$ introduces nonzero off-diagonal elements in A. For $i = n, \ldots, n-b+1$, this fill-in stays within the b bands. For $i \leq n-b$, two undesired nonzero elements are created in row $i-1$ and column $i-1$ outside the b bands. These nonzero elements are immediately chased off to the bottom right corner by applying $n-b-i+1$ Givens rotations, akin to Schwarz band

reduction [102].

When the procedure is completed, the b bands of A are fully populated.

In all examples presented below, we choose the eigenvalues to be uniformly distributed in $[-1, -\mathsf{gap}] \cup [\mathsf{gap}, 1]$. Our results indicate that the performance of our algorithm is robust with respect to the choice of eigenvalue distribution. In particular, the timings stay almost the same when choosing eigenvalues geometrically graded towards the spectral gap.

4.5.2 Results for tridiagonal matrices

In this section we demonstrate the performance of the hQDWH algorithm 4.5 for symmetric tridiagonal matrices. We compare our new algorithm with two existing methods for computing spectral decompositions: the MATLAB function `eig` and MRRR. Moreover, we demonstrate the behavior of the hQDWH algorithm on matrices of varying difficulties.

Example 4.15 (Accuracy versus gap). First we investigate the behavior of the errors for both the hQDWH and the QDWH algorithm with respect to the spectral gap. Using the construction from Section 4.5.1, we consider $10\,000 \times 10\,000$ tridiagonal matrices with eigenvalues in $[-1, -\mathsf{gap}] \cup [\mathsf{gap}, 1]$, where gap varies from 10^{-15} to 10^{-1}. From Figure 4.6 (left), it can be seen that tiny spectral gaps do not have a significant influence on the distance from identity and the trace error for both algorithms. On the other hand, both $e_{\mathrm{SP}}^{\mathcal{H}}$ and e_{SP}^{Q} are sensitive to a decreasing gap, which reflects the ill-conditioning of the spectral projector for small gaps.

Example 4.16 (Accuracy versus the truncation tolerance ϵ). We again consider a tridiagonal matrix $A \in \mathbb{R}^{10\,000 \times 10\,000}$ with the eigenvalues in $[-1, -10^{-4}] \cup [10^{-4}, 1]$. The truncation tolerance ϵ for recompression in the HODLR format is varied in the interval $[10^{-15}, 10^{-5}]$. Figure 4.6 (right) shows the resulting errors in the hQDWH algorithm. As expected, the errors $e_{\mathrm{id}}^{\mathcal{H}}$, $e_{\mathrm{trace}}^{\mathcal{H}}$, $e_{\mathrm{SP}}^{\mathcal{H}}$ increase as ϵ increases. Both $e_{\mathrm{id}}^{\mathcal{H}}$ and $e_{\mathrm{SP}}^{\mathcal{H}}$ grow linearly with respect to ϵ, while $e_{\mathrm{trace}}^{\mathcal{H}}$ appears to be a little more robust.

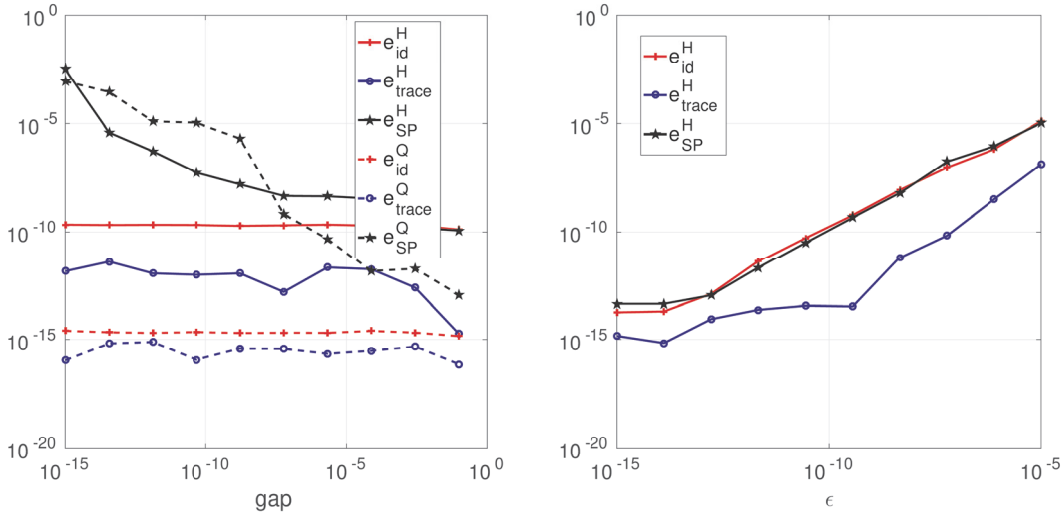

Figure 4.6 – Left (Example 4.15): Comparison of accuracy for the hQDWH and the QDWH algorithm applied to $10\,000 \times 10\,000$ *tridiagonal matrices with varying spectral gap. Right (Example 4.16): Accuracy of the hQDWH algorithm applied to a* $10\,000 \times 10\,000$ *tridiagonal matrix with the spectral gap* 10^{-4} *for different truncation tolerances.*

Example 4.17 (Accuracy for examples from matrix collections). We test the accuracy of the hQDWH algorithm for several matrices from applications, that have also been considered in [86]:

- Matrices from the BCSSTRUC1 set in the Harwell-Boeing Collection [38]. In these examples, a finite element discretization leads to a generalized eigenvalue problem $Kx = \lambda Mx$, where K, M are symmetric positive definite matrices. We consider the equivalent standard eigenvalue problem $L^{-1}KL^{-T}x = \lambda x$, where L denotes the Cholesky factor of M. We shift the matrix $L^{-1}KL^{-T}$ such that approximately half of its spectrum is negative. Finally, the matrix is reduced to a tridiagonal matrix using the MATLAB function `hess`.

- Matrices from UF Sparse Matrix Collection [38]. We consider the symmetric Alemdar and Cannizzo matrices, as well as a matrix from the NASA set. Again, the matrices are shifted such that roughly half of their spectrum is negative, followed by a reduction to tridiagonal form using the MATLAB function `hess`.

Table 5.3 reveals that the hQDWH algorithm yields accurate approximations also for

Table 4.3 – Accuracy of the hQDWH algorithm for tridiagonal matrices from several matrix collections in Example 4.17 in terms of the errors measures $e_{\mathrm{id}}^{\mathcal{H}}$, $e_{\mathrm{trace}}^{\mathcal{H}}$, and $e_{\mathrm{SP}}^{\mathcal{H}}$. The last column reveals the spectral gap associated to the spectral projector $\Pi_{<0}$.

	matrix	n	$e_{\mathrm{id}}^{\mathcal{H}}$	$e_{\mathrm{trace}}^{\mathcal{H}}$	$e_{\mathrm{SP}}^{\mathcal{H}}$	$\|\cdot\|_2$	gap
BCSSTRUC1	bcsst08	1074	10^{-11}	10^{-13}	10^{-9}	$1.68 \cdot 10^{7}$	$7.5 \cdot 10^{-8}$
	bcsst09	1083	10^{-10}	10^{-11}	10^{-8}	$4.29 \cdot 10^{14}$	$3.5 \cdot 10^{-4}$
	bcsst11	1473	10^{-10}	10^{-12}	10^{-7}	$4.75 \cdot 10^{9}$	$4.7 \cdot 10^{-6}$
	Cannizzo matrix	4098	10^{-10}	10^{-10}	10^{-7}	$3.07 \cdot 10^{8}$	$2.4 \cdot 10^{-9}$
	nasa4704	4704	10^{-10}	10^{-12}	10^{-9}	$2.07 \cdot 10^{8}$	$9.8 \cdot 10^{-8}$
	Alemdar matrix	6245	10^{-10}	10^{-11}	10^{-7}	69.5	$7.5 \cdot 10^{-5}$

applications' matrices. More specifically, in all examples, $e_{\mathrm{id}}^{\mathcal{H}}$ and $e_{\mathrm{trace}}^{\mathcal{H}}$ obtain values of order of the truncation tolerance ϵ, while $e_{\mathrm{SP}}^{\mathcal{H}}$ shows dependence on the relative spectral gap.

Example 4.18 (Breakeven point relative to `eig` and MRRR). To compare the computational times of the hQDWH algorithm with `eig`, we consider tridiagonal matrices with eigenvalues contained in $[-1, -\mathsf{gap}] \cup [\mathsf{gap}, 1]$ for various gaps. We have written a `MEX`–function with a direct call to the LAPACK routine `DSTEVD`, to verify that the MATLAB function `eig` indeed is based on the divide-and-conquer algorithm. Table 4.4 shows the resulting breakeven points, that is, the value of n such that hQDWH is faster than `eig` for matrices of size at least n. Not surprisingly, this breakeven point depends on the gap, as ranks are expected to increase as the gap decreases. However, even for $\mathsf{gap} = 10^{-4}$, the hQDWH algorithm becomes faster than `eig` for matrices of moderate size ($n \geq 3250$) and the ranks of the off-diagonal blocks in the HODLR representation of the spectral projector remain reasonably small.

Moreover, we compare the computational time of the hQDWH algorithm with the MRRR algorithm. We have written a `MEX`–function with a direct call to the LAPACK routine `DSTEMR`, which implements the MRRR algorithm. Table 4.4 also shows the resulting breakeven points with respect to MRRR. However, we notice that, compared to breakeven points with respect to `eig`, breakeven points in this case are higher. This is partly due to the fact that, unlike `eig`, MRRR allows to specify the part of spectrum that needs to be computed.

Table 4.4 – Breakeven point of the hQDWH algorithm relative to `eig` *and MRRR for tridiagonal matrices. The last column shows the maximal off-diagonal rank in the output of the hQDWH algorithm.*

gap	eig breakeven point	MRRR breakeven point	max off-diagonal rank
10^{-1}	$n = 2250$	$n = 4700$	18
10^{-2}	$n = 2500$	$n = 5300$	28
10^{-3}	$n = 2750$	$n = 7100$	35
10^{-4}	$n = 3250$	$n = 7500$	37

Example 4.19 (Performance versus n). In this example, we investigate the asymptotic behavior of the hQDWH algorithm, in terms of computational time and memory, for tridiagonal matrices with eigenvalues in $[-1,\ -10^{-6}]\cup[10^{-6},\ 1]$. The left plot of Figure 4.7 indicates that the expected $\mathcal{O}(n\log^2 n)$ computational time is nicely matched. Likewise, the right plot of Figure 4.7 confirms theoretical $\mathcal{O}(n\log n)$ memory requirement. The faster increase for smaller n is because of the fact that the off-diagonal ranks first grow from 30 to 64 until they settle around 64 for sufficiently large n.

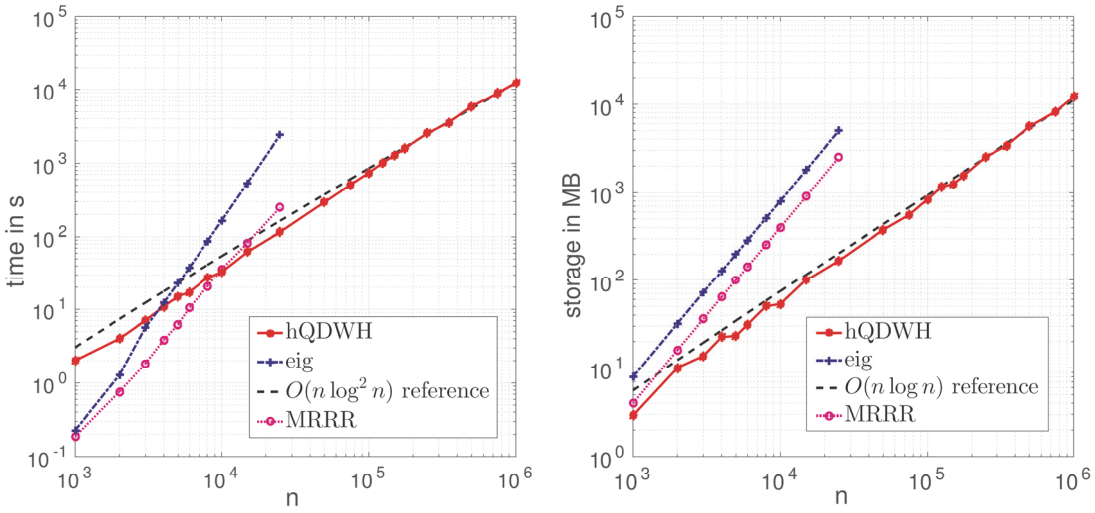

Figure 4.7 – Example 4.19. Performance with respect to n of the hQDWH algorithm, `eig` *and MRRR applied to tridiagonal matrices. Left: Computational time with respect to n. Right: Memory requirements with respect to n.*

Example 4.20 (Performance for 1D Laplace). It is interesting to test the performance of the hQDWH algorithm for matrices for which the spectral gap decreases as n increases. The archetypical example is the (scaled) tridiagonal matrix from the central

difference discretization of the 1D Laplace operator, with eigenvalues $\lambda_k = 2 - 2\cos\frac{k\pi}{n+1}$ for $k = 1, \ldots, n$. The matrix is shifted by 2, such that half of its spectrum is negative and the eigenvalues become equal to $\lambda_k = -2\cos\frac{k\pi}{n+1}$. The spectral gap is given by

$$\mathsf{gap} = \frac{\sin\frac{\pi}{2(n+1)}}{\sin\frac{(n-1)\pi}{2(n+1)}} = \mathcal{O}(1/n).$$

According to Theorem 4.8, the numerical ranks of the off-diagonal blocks depend logarithmically on the spectral gap. Thus, we expect that the hQDWH algorithm requires $\mathcal{O}(n\log^4 n)$ computational time and $\mathcal{O}(n\log^2 n)$ memory for this matrix. Figure 4.8 nicely confirms this theoretical expectation.

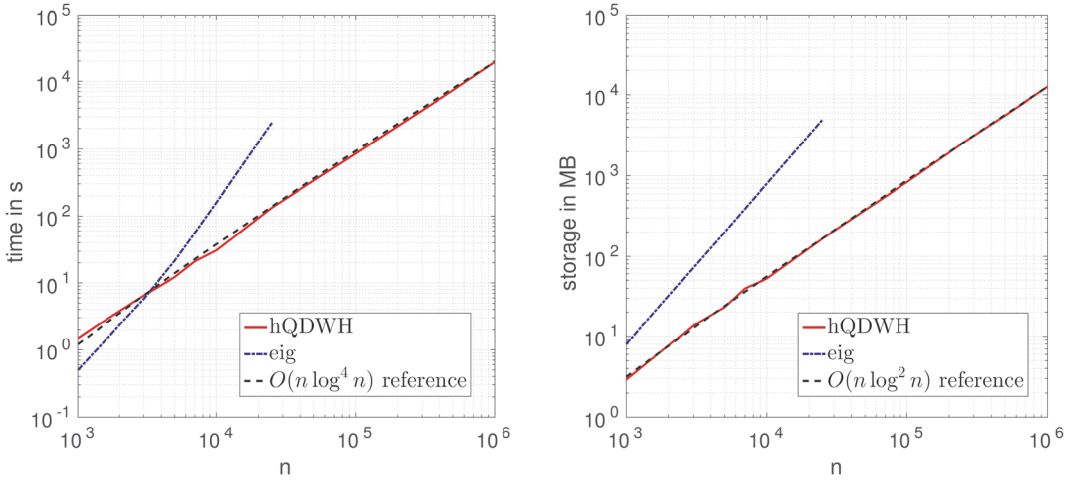

Figure 4.8 – Example 4.20. Performance with respect to n of the hQDWH algorithm and `eig` *for discretized (shifted and scaled) 1D Laplace. Left: Computational time with respect to n. Right: Memory requirements with respect to n.*

4.5.3 Results for banded matrices

In this section we demonstrate the performance of the hQDWII algorithm for banded matrices with bandwidth $b > 1$. More specifically, we test the accuracy, the computational and the storage complexity for various banded matrices, and compare the performance to the MATLAB function `eig`.

Example 4.21 (Accuracy versus gap). Similarly to Example 4.15, we study the impact of the spectral gap on the accuracy of the hQDWH and the QDWH algorithm for banded

matrices. Using once again the construction from Section 4.5.1, we consider $10\,000 \times 10\,000$ banded matrices with bandwidth 8 and eigenvalues in $[-1,\ -\mathsf{gap}] \cup [\mathsf{gap},\ 1]$, where gap varies from 10^{-15} to 10^{-1}. The left plot of Figure 4.9 reconfirms the observations from Example 4.15. More specifically, again we can notice that the errors $e^{\mathcal{H}}_{\mathrm{SP}}$ and e^{Q}_{SP} are more sensitive to a decreasing gap, while tiny spectral gaps do not impact the distance from identity and the trace error for both algorithms.

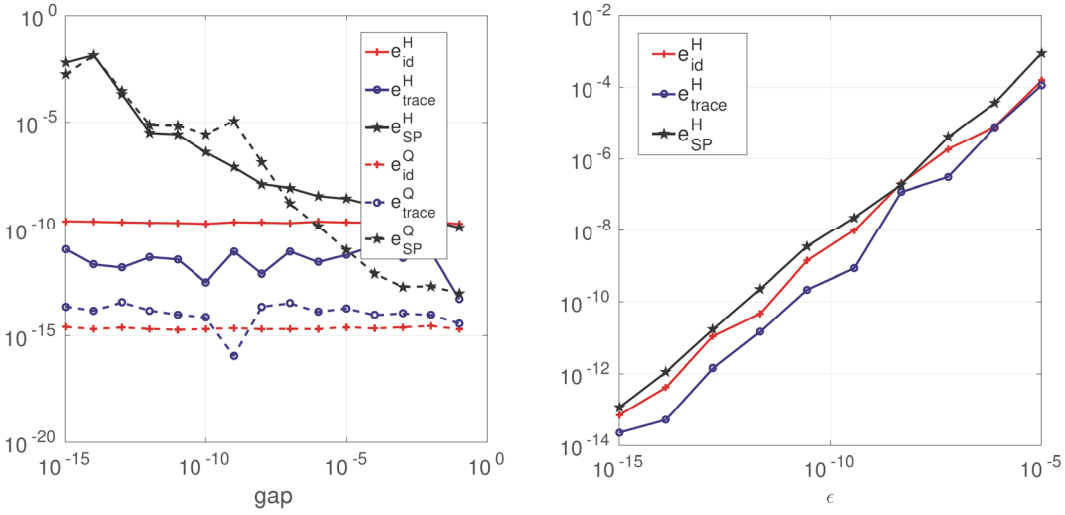

Figure 4.9 – Left (Example 4.21): Comparison of accuracy for the hQDWH and the QDWH algorithm applied to banded matrices with bandwidth 8 with varying spectral gap. Right (Example 4.22): Accuracy of the hQDWH algorithm applied to a banded matrix with bandwidth 8 and the spectral gap 10^{-4}, with respect to different truncation tolerances.

Example 4.22 (Accuracy versus the truncation tolerance ϵ). In this example we investigate the influence of the truncation tolerance ϵ on accuracy of the hQDWH algorithm for an $10\,000 \times 10\,000$ banded matrix with bandwidth $b = 8$ and the eigenvalues contained in $[-1,\ -10^{-4}] \cup [10^{-4},\ 1]$. The right plot of Figure 4.9 presents the resulting errors in the hQDWH algorithm and reconfirms the observations from Example 4.16. In particular, it shows that the errors $e^{\mathcal{H}}_{\mathrm{id}}$, $e^{\mathcal{H}}_{\mathrm{trace}}$, $e^{\mathcal{H}}_{\mathrm{SP}}$ increase linearly with respect to ϵ.

Example 4.23 (Breakeven point relative to `eig`). The aim of this example is to examine when the hQDWH algorithm becomes faster than `eig`, depending on a bandwidth and a spectral gap. Table 4.5 shows when the hQDWH outperforms `eig` for $n \times n$ banded matrices with eigenvalues contained in $[-1,\ -\mathsf{gap}] \cup [\mathsf{gap},\ 1]$ for $\mathsf{gap} = 10^{-1}$ and 10^{-4}. Compared to Table 4.4, the breakeven point is lower for bandwidths $b = 2$ and $b = 4$

than for bandwidth 1 for both considered gaps. These results occur because `eig` needs to perform tridiagonal reduction for matrices with bandwidth $b \geq 2$.

Table 4.5 – Breakeven point of the hQDWH algorithm relative to `eig` applied for banded matrices with various bandwidths and spectral gaps.

gap \ b	2	4	8	16
10^{-1}	$n = 1250$	$n = 1750$	$n = 2500$	$n = 5250$
10^{-4}	$n = 1750$	$n = 2500$	$n = 5000$	$n = 9500$

Example 4.24 (Performance versus n). To test the computational and the storage complexity of the algorithm, we now consider banded matrices with bandwidth 4 and with eigenvalues contained in $[-1,\ -10^{-1}] \cup [10^{-1},\ 1]$. As in Example 4.19, the left plot of Figure 4.10 confirms that the computational time of hQDWH scales like $\mathcal{O}(n \log^2 n)$, while the right plot of Figure 4.10 shows that the memory scales like $\mathcal{O}(n \log n)$, and therefore confirms theoretical complexity. Note that the maximal rank in the off-diagonal blocks of the spectral projector is 66 for $n = 1\,000$, and 71 for $n = 500\,000$.

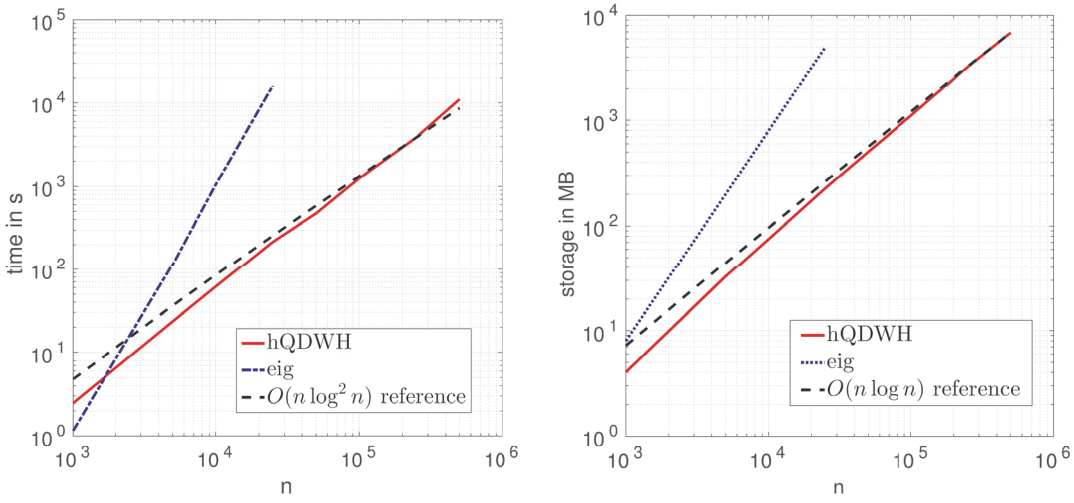

Figure 4.10 – Example 4.24. Performance with respect to n of the hQDWH algorithm and `eig` applied to banded matrices with bandwidth 4 and the spectral gap 10^{-1}. Left: Computational time with respect to n. Right: Memory requirements with respect to n.

Example 4.25 (Performance versus b). To verify the influence of the matrix bandwidth on the performance of our algorithm, we consider $100\,000 \times 100\,000$ banded matrices with eigenvalues contained in $[-1,\ -10^{-6}] \cup [10^{-6},\ 1]$. The left plot of Figure 4.11 clearly

demonstrates that computational time grows quadratically. Additionally, the right plot of Figure 4.11 shows that memory grows linearly with respect to the bandwidth b. Therefore, both theoretical bounds are confirmed.

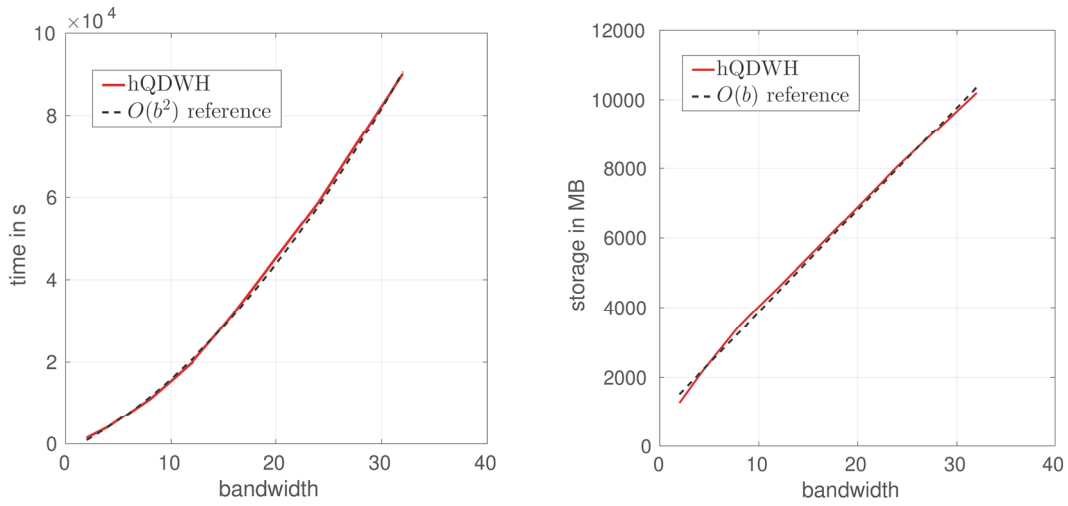

Figure 4.11 – Example 4.25. Performance with respect to bandwidth b of the hQDWH algorithm applied to $100\,000 \times 100\,000$ banded matrices with the spectral gap 10^{-6}. Left: Computational time with respect to b. Right: Memory requirements with respect to b.

4.6 Conclusion

We have developed a fast algorithm for computing spectral projectors of large-scale symmetric banded matrices. For this purpose, we have tailored the ingredients of the QDWH algorithm, such that the overall algorithm has linear-polylogarithmic complexity. This allows us to compute highly accurate approximations to the spectral projector for very large sizes (up to $n = 1\,000\,000$ on a desktop computer) even when the involved spectral gap is small.

The choice of hierarchical low-rank matrix format is critical to the performance of our algorithm. Somewhat surprisingly, we have observed in Chapter 2 that the relatively simple HODLR format outperforms a more general $\mathcal{H}$-matrix format. We have not investigated the implementation of our algorithm in a format with nested low-rank factors, such as HSS matrices. While such a nested format likely lowers asymptotic complexity, it presumably only pays off for larger values of n.

5 A fast spectral divide-and-conquer method for banded matrices

Given a large symmetric banded matrix $A \in \mathbb{R}^{n\times n}$, we consider the computation of its *complete* spectral decomposition

$$A = Q\Lambda Q^T, \quad \Lambda = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n), \tag{5.1}$$

where $\lambda_i, i = 1, \ldots, n$ are the eigenvalues of A and the columns of the orthogonal matrix $Q \in \mathbb{R}^{n\times n}$ the corresponding eigenvectors. This problem has attracted quite some attention from the early days of numerical linear algebra until today, particularly when A is a tridiagonal matrix.

A number of applications give rise to banded eigenvalue problems. For example, they constitute a critical step in solvers for *general* dense symmetric eigenvalue problems. Nearly all existing approaches, with the notable exception of [91], first reduce a given dense symmetric matrix to tridiagonal form. This is followed by a method for determining the spectral decomposition of a tridiagonal matrix, such as the QR algorithm, the classical divide-and-conquer (D&C) method or the MRRR algorithm. All these methods have complexity $\mathcal{O}(n^2)$ or higher; simply because all n eigenvectors are computed and stored explicitly.

As already mentioned in Chapter 4, on a modern computing architecture with a memory hierarchy, it turns out to be advantageous to perform the tridiagonalization based on a successive band reduction [27], with a symmetric banded matrix as an intermediate step [5, 26, 71, 73, 105]. In this context, it would be preferable to design an eigenvalue

solver that works directly with banded matrices, therefore avoiding the reduction from banded to tridiagonal form. We recall that such a possibility has been explored for classical D&C in [4, 72]. However, the proposed methods seem to suffer from numerical instability or an unsatisfactory complexity growth as the bandwidth increases.

In this chapter we propose a new and fast approach for computing the spectral decomposition of a symmetric banded matrix. This is based on the spectral D&C method from [91], which recursively splits the spectrum using invariant subspaces extracted from spectral projectors associated with roughly half of the spectrum. In Chapter 4 we have developed a fast method for approximating such spectral projectors in the HODLR format. However, the extraction of the invariant subspace, requires to determine a basis for the range of the spectral projector. This represents a major challenge. We present an efficient algorithm for computing an orthonormal basis of an invariant subspace in the HODLR format, which heavily exploits properties of spectral projectors. The matrix of eigenvectors is stored implicitly, via orthonormal factors, where each factor is an orthonormal basis for an invariant subspace. Our approach extends to general symmetric HODLR matrices.

Several existing approaches that use hierarchical low-rank formats for the fast solution of eigenvalue problems are based on computing (inexact) LDL^T decompositions in such a format; see [67, Section 13.5] and Chapter 3 for an overview. These decompositions allow to slice the spectrum of a symmetric matrix into smaller chunks and are particularly well suited when only the eigenvalues and a few eigenvectors are needed.

To the best our knowledge, the only existing fast methods suitable for the complete spectral decomposition of a large symmetric matrix are based on variations of the classical D&C method by Cuppen for a symmetric tridiagonal matrix [36]. One recursion of the method divides, after a rank-one perturbation, the matrix into a 2×2 block diagonal matrix. In the conquer phase the rank-one perturbation is incorporated by solving a secular equation for the eigenvalues and applying a Cauchy-like matrix to the matrix of eigenvectors. Gu and Eisenstat [64] not only stabilized Cuppen's method but also observed that the use of the fast multipole method for the Cauchy-like matrix multiplication reduced its complexity to $\mathcal{O}(n^2)$ for computing all eigenvectors. As mentioned in Chapter 3, Vogel et al. [111] extended these ideas beyond tridiagonal matrices, to general symmetric HSS (hierarchically semiseparable) matrices. Moreover, by representing the matrix of eigenvectors in factored form, the overall cost reduces to $\mathcal{O}(n \log^2 n)$. While our work bears similarities with [111], such as the storage of eigenvectors in factored form, it differs

in several key aspects. First, our developments use the HODLR format while [111] uses the HSS format. The latter format stores the low-rank factors of off-diagonal blocks in a nested manner and thus reduces the memory requirements by a factor $\log n$ *if* the involved ranks stay on the same level. However, one may need to work with rather large values of n in order to gain significant computational savings from working with HSS instead of HODLR. A second major difference is that the spectral D&C method used in this work has, despite the similarity in name, little in common with Cuppen's D&C. One advantage of using spectral D&C is that it conveniently allows to compute only parts of the spectrum. A third major difference is that [111] incorporates a perturbation of rank $r > 1$, as it is needed to process matrices of bandwidth larger than one by sequentially splitting it up into r rank-one perturbations. The method presented in this work processes higher ranks directly, avoiding the need for splitting and leveraging the performance of level 3 BLAS operations. While the timings reported in [111] cover matrices of size up to 10 240 and appear to be comparable with the timings presented in this work, our experiments additionally demonstrate that our newly proposed method allows for conveniently dealing with large-scale matrices.

The rest of this chapter is organized as follows. In Section 5.1, we recall the spectral divide-and-conquer algorithm for computing the spectral decomposition of a symmetric matrix, as well as a fast computation of spectral projectors. By extending the newly proposed method for computing a QR decomposition of a HODLR matrix in Chapter 2, we derive a new efficient method for the first iterate of the QDWH algorithm for a symmetric HODLR matrix. In Section 5.2 we discuss the fast extraction of invariant subspaces from a spectral projector given in the HODLR format. Section 5.3 presents the overall spectral D&C algorithm in the HODLR format for computing the spectral decomposition of a symmetric matrix. Numerical experiments are presented in Section 5.4, and conclusion in Section 5.5.

This chapter is largely based on the preprint [108].

5.1 Spectral divide-and-conquer method

In this section we recall the spectral D&C method by Nakatsukasa and Higham [91] for a symmetric $n \times n$ matrix A with spectral decomposition (5.1). We assume that the

eigenvalues are sorted in the ascending order and choose a shift $\mu \in \mathbb{R}$ such that

$$\lambda_1 \leq \cdots \leq \lambda_\nu < \mu < \lambda_{\nu+1} \leq \cdots \leq \lambda_n, \qquad \nu \approx n/2.$$

The relative spectral gap associated with this splitting of eigenvalues is defined as

$$\text{gap} = \frac{\lambda_{\nu+1} - \lambda_\nu}{\lambda_n - \lambda_1}.$$

As mentioned in Section 4.1, the spectral projector associated with the first ν eigenvalues is the orthogonal projector onto the subspace spanned by the corresponding eigenvectors. Given (5.1), it takes the form

$$\Pi_{<\mu} = Q \begin{bmatrix} I_\nu & 0 \\ 0 & 0 \end{bmatrix} Q^T.$$

Note that $\Pi_{<\mu}$ satisfies the following properties:

$$\Pi_{<\mu}^T = \Pi_{<\mu}^2 = \Pi_{<\mu}, \qquad \text{trace}(\Pi_{<\mu}) = \text{rank}(\Pi_{<\mu}) = \nu.$$

The spectral projector associated with the other $n - \nu$ eigenvalues is given by

$$\Pi_{>\mu} = Q \begin{bmatrix} 0 & 0 \\ 0 & I_{n-\nu} \end{bmatrix} Q^T$$

and satisfies analogous properties.

The method from [91] first computes the matrix sign function and then extracts the spectral projectors via the relations

$$\Pi_{<\mu} = \frac{1}{2}(I - \text{sign}(A - \mu I)), \qquad \Pi_{>\mu} = I - \Pi_{<\mu}.$$

The ranges of these spectral projector are invariant subspaces of $A - \mu I$ and, in turn, of A. Letting $Q_{<\mu} \in \mathbb{R}^{n\times\nu}$ and $Q_{>\mu} \in \mathbb{R}^{n\times(n-\nu)}$ denote arbitrary orthonormal bases for $\text{Range}(\Pi_{<\mu})$ and $\text{Range}(\Pi_{>\mu})$, respectively, we therefore obtain

$$\begin{bmatrix} Q_{<\mu} & Q_{>\mu} \end{bmatrix}^T A \begin{bmatrix} Q_{<\mu} & Q_{>\mu} \end{bmatrix} = \begin{bmatrix} A_{<\mu} & 0 \\ 0 & A_{>\mu} \end{bmatrix}, \tag{5.2}$$

where the eigenvalues of $A_{<\mu} = Q_{<\mu}^T A Q_{<\mu}$ are $\lambda_1, \ldots, \lambda_\nu$ and the eigenvalues of $A_{>\mu} = Q_{>\mu}^T A Q_{>\mu}$ are $\lambda_{\nu+1}, \ldots, \lambda_n$. Applying the described procedure recursively to $A_{<\mu}$ and $A_{>\mu}$ leads to Algorithm 5.1. When the size of the matrix is below a user-prescribed minimal size n_{stop}, the recursion is stopped and a standard method for computing spectral decompositions is used, denoted by `eig`.

Algorithm 5.1 Spectral D&C method

Input: Symmetric matrix $A \in \mathbb{R}^{n\times n}$.

Output: Spectral decomposition $A = Q\Lambda Q^T$.

1: **function** $[Q, \Lambda] =$ `sdc(A)`
2: **if** $n \leq n_{\text{stop}}$ **then**
3: Return $[Q, \Lambda] =$ `eig`(A).
4: **else**
5: Choose shift μ.
6: Compute sign function of $A - \mu I$ and extract spectral projectors $\Pi_{<\mu}$ and $\Pi_{>\mu}$.
7: Compute orthonormal bases $Q_{<\mu}, Q_{>\mu}$ of Range$(\Pi_{<\mu})$, Range$(\Pi_{>\mu})$.
8: Compute $A_{<\mu} = Q_{<\mu}^T A Q_{<\mu}$ and $A_{>\mu} = Q_{>\mu}^T A Q_{>\mu}$.
9: Call recursively $[Q_1, \Lambda_1] =$ `sdc`$(A_{<\mu})$ and $[Q_2, \Lambda_2] =$ `sdc`$(A_{>\mu})$.
10: Set $Q \leftarrow \begin{bmatrix} Q_{<\mu}Q_1 & Q_{>\mu}Q_2 \end{bmatrix}$, $\Lambda = \begin{bmatrix} \Lambda_1 & 0 \\ 0 & \Lambda_2 \end{bmatrix}$.
11: **end if**
12: **end function**

In the following sections, we derive ingredients for the efficient implementation of Algorithm 5.1 in the HODLR format.

5.1.1 Computation of spectral projectors in the HODLR format

In Chapter 4 we have introduced a method for computing spectral projectors of banded matrices based on the dynamically weighted Halley iteration from [89, 91]. In this chapter, in order to perform Algorithm 5.1 completely in the HODLR format, we also need a slight variation of that method for dealing with general symmetric HODLR matrices.

The algorithm hQDWH presented in Chapter 4 for banded A is essentially an implementation of the DWH recurrence (4.5) in the HODLR matrix arithmetic, with one major difference. Following [91], the first iteration of hQDWH avoids the computation of the Cholesky factorization for the evaluation of $(I + c_0 X_0^T X_0)^{-1} = (I + c_0/\alpha^2 A^2)^{-1}$ in the first iteration. Instead, a QR decomposition of a $2n \times n$ matrix $\begin{bmatrix} \sqrt{c_0} X_0 \\ I \end{bmatrix}$ is computed, as explained in Section 4.3. This improves numerical stability and allows us to safely

determine spectral projectors even for relatives gaps of order 10^{-16}. In order to complete Algorithm 5.1, one needs to compute spectral projectors of symmetric HODLR matrices as well. For reasons explained in Section 2.2.8, existing algorithms for performing QR decompositions of HODLR matrices come with various drawbacks. However, the newly proposed algorithm for computing QR decompositions presented in Section 2.2.9 can be extended and tailored for computing a QR decomposition of $\begin{bmatrix} \sqrt{c_0}X_0 \\ I \end{bmatrix}$, when X_0 is HODLR. An extension of Algorithm 2.12 is derived in the following section.

QR-based first iteration for HODLR A

Following the method developed in Section 2.2.9 for computing a QR decomposition of a HODLR matrix in terms of a compact WY representation, we derive a new method for performing the first step of the QDWH algorithm. In particular, we recall that a QR-based iteration (4.16) requires the computation of

$$\begin{bmatrix} A \\ I \end{bmatrix} = \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} R, \tag{5.3}$$

where A is a symmetric HODLR matrix.

Given a QR decomposition

$$\begin{bmatrix} A \\ I \end{bmatrix} = Q \begin{bmatrix} R \\ 0 \end{bmatrix} = \left(I - \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} T \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix}^T \right) \begin{bmatrix} R \\ 0 \end{bmatrix}, \tag{5.4}$$

with the orthogonal factor Q given in a compact WY representation, it follows that the factors Q_1 and Q_2 from (5.3) can be obtained as

$$Q_1 = I - Y_1 T Y_1^T, \quad Q_2 = -Y_2 T Y_1^T.$$

The hQR algorithm for computing a QR decomposition of a $n \times n$ HODLR matrix is performed in a recursive block column manner. We proceed analogously for computing (5.3). The main difference occurs when performing operations with block columns of $B = \begin{bmatrix} A \\ I \end{bmatrix}$.

For $A \in \mathcal{H}_{n\times n}(k)$ of level p, associated with the integer partition (2.2), the identity

matrix I in (5.3) is stored in the HODLR format generated by the same integer partition. This implies that I also has 2^p block columns on the lowest level of subdivision. Then B is represented in terms of two HODLR matrices, with the same HODLR structure. Therefore, we can define a block column of B by extracting the corresponding block columns from A and I. In particular, on the lowest level of subdivision in B, a block column consists of a block column on the lowest HODLR level of A appended by a block column on the lowest HODLR level of I. Moreover, the orthogonal factor Q in (5.4) is stored in terms of triangular $n \times n$ HODLR matrices Y_1, Y_2 and T. Thus the computation by block columns of B results in the computation of the corresponding block columns in Y_1, Y_2 and T.

As a result of the structure of I, block columns on the lowest level of subdivision on which we apply Algorithm 2.11, consist of $p + 2$ blocks: a dense matrix (a diagonal block from A), p off-diagonal blocks from A and I, and the identity matrix (a diagonal block from I). Indeed, as discussed in Section 2.2.9, in the computation of the compact WY representation on the lowest HODLR level, sub-block columns consisting of a diagonal block and the lower off-diagonals blocks are needed; see Figure 2.4 for an illustration. The lower off-diagonal blocks in I do not contribute to the compact WY representation, as they are zero prior to the computation. Moreover, the update of the second column is performed analogously to the computations shown on Figure 2.5, also taking into account the triangular structure of the matrix in the lower $n \times n$ block.

As the method for computing (5.4) in terms of the compact WY representation in the HODLR format is an analogous extension of Algorithm 2.12, but requires cumbersome notation, we omit to provide a pseudocode.

General algorithm for spectral projectors

When the starting matrix in Algorithm 5.1 is banded, spectral projectors are computed using the hQDWH algorithm; see Algorithm 4.5.

The parameters α and l_0 needed to start the QDWH algorithm for a HODLR matrix A are computed by applying a few steps of the (inverse) power method to A^2 and A^2/α^2, respectively. The complete algorithm for computing spectral projectors of banded and general symmetric HODLR matrices is provided in Algorithm 5.2.

Algorithm 5.2 hDWH algorithm

Input: Symmetric b-banded $A \in \mathbb{R}^{n\times n}$ or symmetric $A \in \mathcal{H}_{n\times n}(k)$, truncation tolerance $\epsilon > 0$, stopping tolerance $\varepsilon > 0$.

Output: Approximate spectral projectors $\Pi_{<0}$ and $\Pi_{>0}$ in the HODLR format.

if A is banded **then**
  Compute $\Pi_{<0}$ and $\Pi_{>0}$ using the hQDWH algorithm (see Section 4.4.)
else
  Compute initial parameters $\alpha \gtrsim \|A\|_2$ via power iteration on A^2 and $l_0 \lesssim \sigma_{\min}(A/\alpha)$ via inverse power iteration on A^2/α^2.
  $X_0 = A/\alpha$.
  $k = 0$.
  while $|1 - l_k| > \varepsilon$ **do**
    Compute a_k, b_k, c_k according to the recurrence (4.14).
    if $k = 0$ **then**
      Compute $\begin{bmatrix}\sqrt{c_0}X_0 \\ I\end{bmatrix} = \left(I - \begin{bmatrix}V_1 \\ V_2\end{bmatrix} *_{\mathcal{H}} T *_{\mathcal{H}} \begin{bmatrix}V_1 \\ V_2\end{bmatrix}^T\right) *_{\mathcal{H}} R$ using Algorithm 2.12.
      Compute $Q_1 = I - V_1 *_{\mathcal{H}} T *_{\mathcal{H}} V_1^T$ and $Q_2 = -V_2 *_{\mathcal{H}} T *_{\mathcal{H}} V_1^T$.
      Compute $X_1 = \frac{b_0}{c_0}X_0 +_{\mathcal{H}} \frac{1}{\sqrt{c_0}}\left(a_0 - \frac{b_0}{c_0}\right) Q_1 *_{\mathcal{H}} Q_2^T$.
    else
      Compute $W_k = \texttt{hchol}(I + c_k X_k^T *_{\mathcal{H}} X_k)$.
      Solve upper triangular system $Y_k W_k = X_k$ in the HODLR format using a variant of Algorithm 2.5.
      Solve lower triangular system $V_k W_k^T = Y_k$ in the HODLR format using Algorithm 2.5.
      Compute $X_{k+1} = \frac{b_k}{c_k}X_k +_{\mathcal{H}} \left(a_k - \frac{b_k}{c_k}\right) V_k$.
    end if
    $k = k + 1$.
    Compute l_k according to the recurrence (4.15).
  end while
  Set $U = X_k$.
  Return $\Pi_{<0} = \frac{1}{2}(I - U)$ and $\Pi_{>0} = \frac{1}{2}(I + U)$.
end if

Taking into account the complexity of operations used in Algorithm 5.2, and assuming that a constant number of iterations is needed and that the HOLDR ranks of all intermediate quantities are bounded by k, the complexity of Algorithm 5.2 is $\mathcal{O}(k^2 n \log^2 n)$.

Remark 5.1. As a result of the new algorithm for computing the QR-based iteration for general symmetric HODLR matrices, the DWH iteration can be implemented in the HODLR format using only the QR-based iteration. However, because of a lower flop count,

and already demonstrated numerical accuracy, we always perform QR-based iteration only in the first step, and then switch to the Cholesky-based iteration.

5.2 Computation of invariant subspace basis in the HODLR format

This section addresses the efficient extraction of a basis for the range of a spectral projector $\Pi_{<\mu}$ given in the HODLR format.

Assuming that $\operatorname{rank}(\Pi_{<\mu}) = \nu$, the most straightforward approach to obtain a basis for $\operatorname{Range}(\Pi_{<\mu})$ is to simply take its first ν columns. Numerically, this turns out to be a terrible idea, especially when A is banded.

Example 5.2. Let n be even and let $A \in \mathbb{R}^{n\times n}$ be a symmetric banded matrix with bandwidth b and eigenvalues distributed uniformly in $[-1,\ -10^{-1}] \cup [10^{-1},\ 1]$. In particular, $\operatorname{rank}(\Pi_{<0}) = n/2$. Figure 5.1 shows that the condition number of the first $n/2$ columns of $\Pi_{<0}$ grows dramatically as n increases. By computing a QR decomposition of these columns, we obtain an orthonormal basis $Q_1 \in \mathbb{R}^{n\times n/2}$. This basis has perfect condition number but, as Table 5.1 shows, it represents a rather poor approximation of $\operatorname{Range}(\Pi_{<0})$.

Table 5.1 – Example 5.2. Angles (in radians) between $\operatorname{Range}(\Pi_{<0})$ *and* $\operatorname{Range}(Q_1)$*, with* Q_1 *an orthonormal basis for* $\operatorname{Range}(\Pi_{<0}(:, 1:\frac{n}{2}))$.

n	$\measuredangle(\operatorname{Range}(\Pi_{<0}), \operatorname{Range}(Q_1)))$
64	$4.4916e-02$
256	$1.5692e+00$
1024	$1.5700e+00$
4096	$1.5707e+00$

There exist a number of approaches that potentially avoid the problems observed in Example 5.2, such as a QR factorization with pivoting [60, Chapter 5.4] for $\Pi_{<0}$. None of these approaches has been realized in the HODLR format. In fact, techniques like pivoting across blocks appear to be incompatible with the format.

In the following, we develop a new method for computing a basis for $\operatorname{Range}(\Pi_{<\mu})$ in the HODLR format, which consists of two steps:

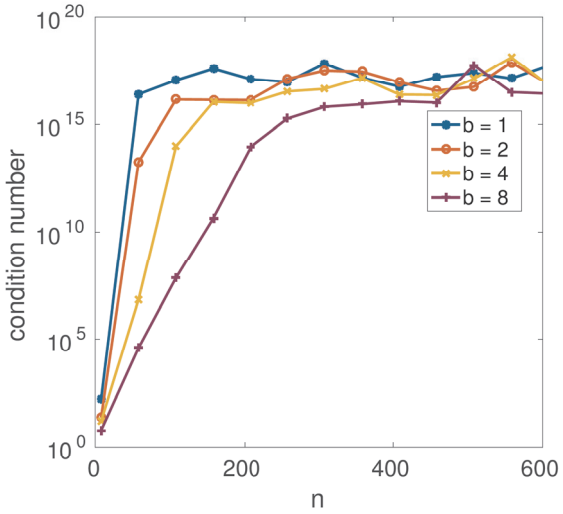

Figure 5.1 – Example 5.2. The condition number of the first $n/2$ columns of the spectral projector $\Pi_{<0}$ for matrices with bandwidths $b = 1, 2, 4, 8$.

1. We first determine a set of well-conditioned columns of $\Pi_{<\mu}$ by performing a Cholesky factorization with *local* pivoting. As we will see below, the number of obtained columns is generally smaller than ν, but not much smaller.

2. A randomized algorithm is applied to complete the columns to a basis of $\mathrm{Range}(\Pi_{<\mu})$.

5.2.1 Column selection by block Cholesky with local pivoting

The spectral projector $\Pi_{<\mu}$ is not only symmetric positive semidefinite but it is also idempotent. The (pivoted) Cholesky factorization of such matrices has particular properties.

Theorem 5.3 ([76, Theorem 10.9])**.** *Let $B \in \mathbb{R}^{n\times n}$ be a symmetric positive semidefinite matrix of rank r. Then there is a permutation matrix P such that P^TBP admits a Cholesky factorization:*

$$P^TBP = R^TR, \quad R = \begin{bmatrix} R_1 & R_2 \\ 0 & 0 \end{bmatrix},$$

where R_1 is a $r \times r$ upper triangular matrix with positive diagonal elements.

Note that, by the invertibility of R_1, the first r columns of BP form a basis for $\mathrm{Range}(B)$.

The same holds for $[R_1 \quad R_2]^T$. The latter turns out to be orthonormal if B is idempotent.

Lemma 5.4 ([88, Corollary 1.2.]). *Suppose, in addition to the hypotheses of Theorem 5.3, that $B^2 = B$. Then*

$$[R_1 \quad R_2] \begin{bmatrix} R_1^T \\ R_2^T \end{bmatrix} = I_r.$$

The algorithm described in [76, Chapter 10] for realizing Theorem 5.3 chooses the maximal diagonal element as the pivot in every step of the standard Cholesky factorization algorithm. In turn, the diagonal elements of the Cholesky factor are monotonically decreasing and it is safe to decide which ones are considered zero numerically. Unfortunately, this algorithm, which will be denoted by `cholp` in the following, cannot be applied to $\Pi_{<\mu}$ because the diagonal pivoting strategy destroys the HODLR format. Instead, we use `cholp` only for the (dense) diagonal blocks of $\Pi_{<\mu}$.

To illustrate the idea of our algorithm, we first consider a general symmetric positive semidefinite HODLR matrix M of level 1, which takes the form

$$M = \begin{bmatrix} M_{11} & U_1 V_2^T \\ V_2 U_1^T & M_{22} \end{bmatrix}$$

with dense diagonal blocks M_{11}, M_{22}. Applying `cholp` to M_{11} gives a decomposition $P_1^T M_{11} P_1 = R_{11}^T R_{11}$, with the diagonal elements of R_{11} decreasing monotonically. As M, and in turn also M_{11}, will be chosen as a principal submatrix of $\Pi_{<\mu}$, Lemma 5.4 implies that $\|R_{11}\|_2 \leq 1$. In particular, the diagonal elements of R_{11} are bounded by 1. Let s_1 denote the number of diagonal elements not smaller than a prescribed threshold δ. As will be shown in Lemma 5.5 below, choosing δ sufficiently close to 1 ensures that $R_{11}(1:s_1, 1:s_1)$ is well-conditioned. Letting π_1 denote the permutation associated with P_1 and setting $C_1 = \pi_1(1:s_1)$, we have

$$M_{11}(C_1, C_1) = R_{11}(1:s, 1:s)^T R_{11}(1:s, 1:s).$$

The Schur complement of this matrix in M (without considering the rows and columns neglected in the first block) is given by

$$S = M_{22} - V_2 U_1(C_1, :)^T M_{11}(C_1, C_1)^{-1} U_1(C_1, :) V_2^T = M_{22} - \tilde{R}_{12}^T \tilde{R}_{12}, \tag{5.5}$$

where the rank of $\tilde{R}_{12} := R_{11}(1:s_1, 1:s_1)^{-T} U_1(C_1, :) V_2^T$ is not larger than the rank of

$U_1 V_2^T$. We again apply `cholp` to S and only retain diagonal elements of the Cholesky factor R_{22} larger or equal than δ. Letting C_2 denote the corresponding indices and setting $s_2 = |C_2|$, $C = C_1 \cup (n_1 + C_2)$, where n_1 is the size of M_{11}, we obtain the factorization

$$M(C,C) = \tilde{R}^T \tilde{R} \quad \text{with} \quad \tilde{R} = \begin{bmatrix} R_{11}(1:s_1, 1:s_1) & \tilde{R}_{12}(:, C_2) \\ 0 & R_{22}(1:s_2, 1:s_2) \end{bmatrix}.$$

For a general HODLR matrix, we proceed recursively in an analogous fashion, with the difference that we now form submatrices of HODLR matrices (see Section 2.2.7) and the operations in (5.5) are executed in the HODLR arithmetic.

The procedure described above leads to Algorithm 5.3. Based on the complexity of operations stated in Table 2.6, the cost of the algorithm applied to an $n \times n$ spectral projector $\Pi_{<\mu} \in \mathcal{H}_{n\times n}(k)$ is $\mathcal{O}(k^2 n \log^2 n)$. The linear system in Line 9 is solved as explained in Section 2.2.5. In Line 10 we update a HODLR matrix with a matrix given by its low-rank representation. This operation essentially corresponds to the addition of two HODLR matrices; see Section 2.2.3. In Line 10 we also enforce symmetry in the Schur complement.

Algorithm 5.3 Incomplete Cholesky factorization with local pivoting for symmetric positive semidefinite HODLR matrices

Input: Positive semidefinite HODLR matrix $M \in \mathcal{H}_{n\times n}(k)$ of level p, tolerance $\delta > 0$.
Output: Indices $C \subset [1, n]$ and upper triangular HODLR matrix $\tilde{R}$ such that $M(C,C) \approx \tilde{R}^T \tilde{R}$, with $\tilde{r}_{ii} \geq \delta$ for $i = 1, \ldots, |C|$.

1: **function** $[C, \tilde{R}] =$ `hcholp_inc(M)`
2: **if** $p = 0$ **then**
3: Compute $[R, \pi] =$ `cholp`(M) such that $M(\pi, \pi) = R^T R$.
4: Set s such that $r_{11} \geq \delta, \ldots, r_{ss} \geq \delta$ and $r_{s+1,s+1} < \delta$ (or $s = n$).
5: Return $C = \pi(1:s)$ and $\tilde{R} = R(1:s, 1:s)$.
6: **else**
7: Partition $M = \begin{bmatrix} M_{11} & U_1 V_2^T \\ V_2 U_1^T & M_{22} \end{bmatrix}$.
8: Call recursively $[C_1, \tilde{R}_{11}] =$ `hcholp_inc`(M_{11}).
9: Compute $\tilde{U}_1 = \tilde{R}_{11}^{-T} U_1(C_1, :)$.
10: Compute $S = M_{22} -_{\mathcal{H}} V_2 \tilde{U}_1^T \tilde{U}_1 V_2^T$.
11: Call recursively $[C_2, \tilde{R}_{22}] =$ `hcholp_inc`(S).
12: Return $C = C_1 \cup (n_1 + C_2)$ and HODLR matrix $\tilde{R} = \begin{bmatrix} \tilde{R}_{11} & \tilde{U}_1 V_2(C_2, :)^T \\ 0 & \tilde{R}_{22} \end{bmatrix}$.
13: **end if**
14: **end function**

Analysis of Algorithm 5.3

The indices C selected by Algorithm 5.3 applied to $\Pi_{<\mu}$ need to attain two goals:

1. $\Pi_{<\mu}(:, C)$ has moderate condition number;
2. $|C|$ is not much smaller than the rank of $\Pi_{<\mu}$.

In the following analysis, we show that the first goal is met when choosing δ sufficiently close to 1. The attainment of the second goal is demonstrated by the numerical experiments in Section 5.4.

Our analysis needs to take into account that Algorithm 5.3 is affected by an error due to the truncation in the HODLR arithmetic. On the one hand, the input matrix, the spectral projector $\Pi_{<\mu}$ computed by Algorithm 5.2, is not exactly idempotent:

$$\Pi_{<\mu}^2 = \Pi_{<\mu} + F, \tag{5.6}$$

with a symmetric perturbation matrix F of small norm. On the other hand, the incomplete Cholesky factor $\tilde{R}$ returned by Algorithm 5.3 is inexact as well:

$$\Pi_{<\mu}(C, C) = \tilde{R}^T \tilde{R} + E, \tag{5.7}$$

with another symmetric perturbation matrix E of small norm. For a symmetric matrix $\Pi_{<\mu}$ satisfying (5.6), Theorem 2.1 in [88] shows that

$$\|\Pi_{<\mu}\|_2 \leq 1 + \|F\|_2. \tag{5.8}$$

The following lemma establishes a bound on the norm of the inverse of $\Pi_{<\mu}(C, C)$.

Lemma 5.5. *With the notation introduced above, set* $\varepsilon_{\mathcal{H}} = \|E\|_2 + \|F\|_2$ *and* $r = |C|$, *and suppose that* $1 - \delta^2 + \varepsilon_{\mathcal{H}} < 1/r$. *Then* $\|\Pi_{<\mu}(C, C)^{-1}\|_2 \leq \frac{1}{r} \frac{1}{\delta^2 - 1 + 1/r - \varepsilon_{\mathcal{H}}}$.

Proof. Using (5.7) and (5.8), we obtain

$$\|\tilde{R}^T \tilde{R}\|_2 \leq \|\Pi_{<\mu}(C, C)\|_2 + \|E\|_2 \leq \|\Pi_{<\mu}\|_2 + \|E\|_2 \leq 1 + \varepsilon_{\mathcal{H}}.$$

We now decompose $\tilde{R} = D + T$, such that D is diagonal with $d_{ii} = \tilde{r}_{ii} \geq \delta$ and T is

strictly upper triangular. Then

$$\|D^2 + T^T D + DT + T^T T\|_2 \leq 1 + \varepsilon_{\mathcal{H}}.$$

Because the matrix on the left is symmetric, this implies

$$\lambda_{\max}(D^2 + T^T D + DT + T^T T) \leq 1 + \varepsilon_{\mathcal{H}} \;\Rightarrow\; \lambda_{\max}(T^T D + DT + T^T T) \leq 1 - \delta^2 + \varepsilon_{\mathcal{H}}.$$

On the other hand,

$$\begin{aligned}\lambda_{\min}(T^T D + DT + T^T T) &\geq \lambda_{\min}(T^T D + DT) \\ &\geq -(r-1)\lambda_{\max}(T^T D + DT) \geq -(r-1)(1 - \delta^2 + \varepsilon_{\mathcal{H}}),\end{aligned}$$

where the second inequality uses that the trace of $T^T D + DT$ is zero and hence its eigenvalues sum up to zero. In summary,

$$\|T^T D + DT + T^T T\|_2 \leq (r-1)(1 - \delta^2 + \varepsilon_{\mathcal{H}}),$$

and $\Pi_{<\mu}(C, C) = D^2 + \tilde{E}$ with $\|\tilde{E}\|_2 \leq (r-1)(1-\delta^2) + r\varepsilon_{\mathcal{H}}$. This completes the proof because

$$\begin{aligned}\|\Pi_{<\mu}(C, C)^{-1}\|_2 &\leq \|D^{-2}\|_2 \|(I + D^{-2}\tilde{E})^{-1}\|_2 \leq \frac{1}{\delta^2 - (r-1)(1-\delta^2) - r\varepsilon_{\mathcal{H}}} \\ &= \frac{1}{r}\frac{1}{\delta^2 - 1 + 1/r - \varepsilon_{\mathcal{H}}},\end{aligned}$$

where the inverse exists under the conditions of the lemma. □

The following theorem shows that the columns $\Pi_{<\mu}(:, C)$ selected by Algorithm 5.3 have an excellent condition number if δ is sufficiently close to 1 and the perturbations introduced by the HODLR arithmetic remain small.

Theorem 5.6. *Let C denote the set of r indices returned by Algorithm 5.3 and suppose that the conditions* (5.6) *and* (5.7) *as well as the condition of Lemma 5.5 are satisfied. Then it holds for the* 2*-norm condition number of $\Pi_{<\mu}(:, C)$ that*

$$\kappa(\Pi_{<\mu}(:, C)) \leq \frac{1}{r}\frac{1 + \varepsilon_{\mathcal{H}}}{\delta^2 - 1 + 1/r - \varepsilon_{\mathcal{H}}} = 1 + 2r(1-\delta) + \mathcal{O}((1-\delta)^2 + \varepsilon_{\mathcal{H}}).$$

Proof. By definition, $\kappa(\Pi_{<\mu}(:,C)) = \|\Pi_{<\mu}(:,C)\|_2 \|\Pi_{<\mu}(:,C)^\dagger\|_2$. From (5.8) we get

$$\|\Pi_{<\mu}(:,C)\|_2 \leq \|\Pi_\mu\|_2 \leq 1 + \|F\|_2. \tag{5.9}$$

To bound the second factor, we note that $\|\Pi_{<\mu}(:,C)^\dagger\|_2 \leq \|\Pi_{<\mu}(C,C)^{-1}\|_2$ and apply Lemma 5.5. Using the two bounds, the statement follows. □

The condition of Lemma 5.5, $1 - \delta^2 \lesssim 1/r$, requires δ to be very close to 1. We conjecture that this condition can be improved to a distance that is proportional to $1/\log_2 r$ or even a constant independent of r. The latter is what we observe in the numerical experiments; choosing δ constant and letting r grow does not lead to a deterioration of the condition number.

5.2.2 Range correction

As earlier, let C denote a set of indices obtained by Algorithm 5.3 for a threshold δ, and $r = |C|$. We recall that the dimension of the column space of $\Pi_{<\mu}$ can be easily computed knowing that $\operatorname{trace}(\Pi_{<\mu}) = \operatorname{rank}(\Pi_{<\mu}) = \nu$. If $r = \nu$, then it only remains to perform the orthogonalization to get an orthonormal basis of $\operatorname{Range}(\Pi_{<\mu})$. However, depending on the choice of δ, in practice it can occur that the cardinality of C is smaller than ν, which implies that the selected columns cannot span the column space of $\Pi_{<\mu}$. In this case additional vectors need to be computed to get a complete orthonormal basis for $\operatorname{Range}(\Pi_{<\mu})$.

An orthonormal basis for $\operatorname{Range}(\Pi_{<\mu}(:,C))$ in the HODLR format can be computed using Lintner's QR decomposition [83], reviewed in Section 2.2.8. The basis can be obtained by solving an upper triangular HODLR system, and is given as

$$\Pi_{<\mu}(:,C) *_{\mathcal{H}} \tilde{R}^{-1}. \tag{5.10}$$

The biggest disadvantage of this method is the loss of orthogonality in badly conditioned problems, caused by squaring of the condition number when computing $\tilde{R}$. However, choosing only well-conditioned subset of columns of $\Pi_{<\mu}$ allows us to avoid dealing with badly conditioned problems, and thus prevents potential loss of orthogonality in (5.10).

In case $r < \nu$, we complete the basis (5.10) to an orthonormal basis for $\mathrm{Range}(\Pi_{<\mu})$ by computing an orthonormal basis of the orthogonal complement of $\mathrm{Range}(\Pi_{<\mu}(:, C))$ in $\mathrm{Range}(\Pi_{<\mu})$. First we detect the orthogonal complement of $\mathrm{Range}(\Pi_{<\mu}(:, C))$ in $\mathrm{Range}(\Pi_{<\mu})$.

Lemma 5.7. *If* $(\mathrm{Range}(\Pi_{<\mu}(:, C)))^{\perp}$ *is the orthogonal complement of* $\mathrm{Range}(\Pi_{<\mu}(:, C))$*, then*

$$R^{\perp}_{\Pi_{<\mu},C} := (\mathrm{Range}(\Pi_{<\mu}(:, C)))^{\perp} \cap \mathrm{Range}(\Pi_{<\mu})$$

is the orthogonal complement of $\mathrm{Range}(\Pi_{<\mu}(:, C))$ *in the vector space* $\mathrm{Range}(\Pi_{<\mu})$*. Moreover,* $\dim(R^{\perp}_{\Pi_{<\mu},C}) = \mathrm{rank}(\Pi_{<\mu}) - r$*.*

Proof. The statements follow directly from the definition of $R^{\perp}_{\Pi_{<\mu},C}$. □

Using (5.10) we construct an orthogonal projector

$$P_{C^{\perp}} = I - \Pi_{<\mu}(:, C) *_{\mathcal{H}} \tilde{R}^{-1} *_{\mathcal{H}} (\Pi_{<\mu}(:, C) *_{\mathcal{H}} \tilde{R}^{-1})^{T} \tag{5.11}$$

onto $(\mathrm{Range}(\Pi_{<\mu}(:, C)))^{\perp}$. From (5.10) it readily follows that $\mathrm{Range}(P_{C^{\perp}}\Pi_{<\mu}) = R^{\perp}_{\Pi_{<\mu},C}$. Thus computing an orthonormal basis for $P_{C^{\perp}}\Pi_{<\mu}$ allows us to obtain a complete orthonormal basis for $\mathrm{Range}(\Pi_{<\mu})$.

To this end, we employ a randomized algorithm [74] to compute an orthonormal basis of dimension $\nu - r$ for $\mathrm{Range}(P_{C^{\perp}}\Pi_{<\mu})$:

1. multiply $P_{C^{\perp}}\Pi_{<\mu}$ with a random matrix $X \in \mathbb{R}^{n\times(\nu-r+s)}$, where s is an oversampling parameter;
2. compute its economic QR decomposition with pivoting.

As singular values of $\Pi_{<\mu}$ are either unity or zero, the multiplication with the orthogonal projector $P_{C^{\perp}}$, generated by the linearly independent columns C, gives a matrix whose singular values are well-separated as well. In particular, the resulting matrix has $\nu - r$ singular values equal to 1, and the others equal to zero. Indeed, in the exact arithmetics $P_{C^{\perp}}\Pi_{<\mu}$ has the exact rank $\nu - r$, and then oversampling is not required [74]. However,

because of the use of formatted arithmetics, we employ a small oversampling parameter s to improve the accuracy. As only $\nu - r$ columns are required to complete the basis for Range($\Pi_{<\mu}$), after computing an economic QR decomposition with pivoting in step 2, we keep only the first $\nu - r$ columns of the orthonormal factor.

The pseudocode for computing a complete orthonormal basis for Range($\Pi_{<\mu}$) is presented in Algorithm 5.4. Note that $\Pi_{<\mu}(:, C)$ is a rectangular HODLR matrix, obtained by extracting columns with indices C of a HODLR matrix, as explained in Section 2.2.7. This implies that the complexity of operations stated in Table 2.6 carries over for the operations involving HODLR matrices in Algorithm 5.4. The complexity of the algorithm also depends on the number of the missing basis vectors. However, in our experiments we observe that $\nu - r$ is very small with respect to ν and n for the choice of parameter δ used, which makes the cost of operations in Line 3 and Line 4 negligible. In the setup when $\nu \approx n/2$, the overall complexity of Algorithm 5.4 is governed by solving a triangular system in Line 5 or Line 7, i.e. the complexity of the algorithm is $\mathcal{O}(k^2 n \log^2 n)$.

Algorithm 5.4 Computation of a complete orthonormal basis for Range($\Pi_{<\mu}$)

Input: Spectral projector $\Pi_{<\mu} \in \mathbb{R}^{n\times n}$ in the HODLR format with $\text{rank}(\Pi_{<\mu}) = \nu$, column indices C and the Cholesky factor $\tilde{R}$ returned by Algorithm 5.3, an oversampling parameter s.
Output: Orthonormal matrix $Q_{<\mu} \in \mathbb{R}^{n\times\nu}$ in the HODLR format such that $\text{Range}(Q_{<\mu}) = \text{Range}(\Pi_{<\mu})$.

1: **if** $|C| < \nu$ **then**
2: Generate a random matrix $X \in \mathbb{R}^{n\times(\nu-r+s)}$, for $r = |C|$.
3: $Z = \Pi_{<\mu}X - \Pi_{<\mu}(:, C)(\tilde{R}^{-1}(\tilde{R}^{-T}(\Pi_{<\mu}(C, :)(\Pi_{<\mu}X))))$.
4: Compute $[Q_c, \sim, \sim] = \texttt{qr}(Z, 0)$.
5: Return $Q_{<\mu} = [\Pi_{<\mu}(:, C) *_{\mathcal{H}} \tilde{R}^{-1} \quad Q_c(:, 1 : \nu - r)]$.
6: **else**
7: Return $Q_{<\mu} = [\Pi_{<\mu}(:, C) *_{\mathcal{H}} \tilde{R}^{-1}]$.
8: **end if**

Storing additional columns

When the range correction is performed, we additionally need to store the tall-and-skinny matrix Q_c from Algorithm 5.4. The idea is to incorporate columns of Q_c into an existing HODLR matrix $\Pi_{<\mu}(:, C) *_{\mathcal{H}} \tilde{R}^{-1}$ of size $n \times r$ to get a HODLR matrix of size $n \times \nu$. More specifically, we append $\nu - r$ columns after the last column of $\Pi_{<\mu}(:, C) *_{\mathcal{H}} \tilde{R}^{-1}$, by

enlarging all blocks of $\Pi_{<\mu}(:, C) *_{\mathcal{H}} \tilde{R}^{-1}$ that contain the last column. Recompression is performed when updating the off-diagonal blocks. It is expected that the off-diagonal ranks in the updated blocks grow. However, numerical experiments in Section 5.4 demonstrate that the increase is not significant.

5.3 Divide-and-conquer method in the HODLR format

In this section we give the overall spectral divide-and-conquer method for computing the eigenvalue decomposition of a symmetric banded matrix $A \in \mathbb{R}^{n\times n}$. For completeness, we also include a pseudocode given in Algorithm 5.5. In the following we discuss several details related to its implementation and provide the structure of the eigenvectors matrix.

Algorithm 5.5 Spectral divide-and-conquer algorithm in the HODLR format (hSDC)
Input: A symmetric banded or HODLR matrix $A \in \mathbb{R}^{n\times n}$.
Output: A structured matrix Q containing the eigenvectors of A and a diagonal matrix Λ containing the eigenvalues of A.

1: **function** $[Q, \Lambda] = \texttt{hsdc}(A)$
2: **if** $n \leq n_{\text{stop}}$ **then**
3: $[Q, \Lambda] = \texttt{eig}(A)$.
4: **else**
5: Compute $\mu = \texttt{median}(\text{diag}(A))$.
6: Compute spectral projectors $\Pi_{<\mu}$ and $\Pi_{>\mu}$ in the HODLR format by applying Algorithm 5.2 to $A - \mu I$.
7: Compute column indices $C_{<\mu}$ and $C_{>\mu}$ by applying Algorithm 5.3 to $\Pi_{<\mu}$ and $\Pi_{>\mu}$, respectively.
8: Compute $Q_{<\mu}$ and $Q_{>\mu}$ by applying Algorithm 5.4 to $\Pi_{<\mu}, C_{<\mu}$, and $\Pi_{>\mu}, C_{>\mu}$.
9: Form $A_{<\mu} = Q_{<\mu}^T *_{\mathcal{H}} A *_{\mathcal{H}} Q_{<\mu}$ and $A_{>\mu} = Q_{>\mu}^T *_{\mathcal{H}} A *_{\mathcal{H}} Q_{>\mu}$.
10: Call recursively $[Q_1, \Lambda_1] = \texttt{hsdc}(A_{<\mu})$ and $[Q_2, \Lambda_2] = \texttt{hsdc}(A_{>\mu})$.
11: Set $Q \leftarrow \begin{bmatrix} Q_{<\mu} & Q_{>\mu} \end{bmatrix} *_{\mathcal{H}} \begin{bmatrix} Q_1 & 0 \\ 0 & Q_2 \end{bmatrix}$ and $\Lambda = \begin{bmatrix} \Lambda_1 & 0 \\ 0 & \Lambda_2 \end{bmatrix}$.
12: **end if**
13: end function

5.3.1 Computing the shift

The purpose of computing shift μ is to split a problem of size n into a two smaller subproblems of roughly the same size. In this work, the computation of a shift is performed by computing the median of $\text{diag}(A)$, as proposed in [91]. Although this way

of estimating the median of eigenvalues may not be optimal, it is a cheap method and gives reasonably good results. For more details regarding the shift computation, we refer the reader to a discussion in [91]. Moreover, we note that it remains an open problem to develop a better strategy for splitting the spectrum.

5.3.2 Terminating the recursion

The recursion in Algorithm 5.5 is stopped when the matrix attains the minimal prescribed size n_{stop}. The final step of diagonalization is performed by using MATLAB built-in function `eig`. In a practical implementation, we set n_{stop} depending on the breakeven point of the hQDWH algorithm relative to `eig` presented in Section 4.5.

5.3.3 Matrix of eigenvectors

For simplicity, without loss of generality we assume that for size of a given matrix A holds $n = 2^p n_{\text{stop}}$, for $p \in \mathbb{N}$. We say that Algorithm 5.5 performed a level l divide step, with $0 \leq l < p$, if all matrices of size $n/2^l$ had been subdivided.

The eigenvectors matrix is given as an implicit product of orthonormal HODLR matrices. After level l divide step of Algorithm 5.5, structured matrix Q has the form

$$Q = Q^{(0)} *_{\mathcal{H}} Q^{(1)} *_{\mathcal{H}} \cdots *_{\mathcal{H}} Q^{(l)}.$$

$Q^{(i)} \in \mathbb{R}^{n \times n}$, $0 \leq i \leq l$, are block-diagonal matrices with 2^i diagonal blocks, where each diagonal block is an orthogonal matrix of the form $[H_1\, H_2]$, with H_1, H_2 orthonormal HODLR matrices computed in Line 8 of Algorithm 5.5. The computation of the eigenvectors matrix is completed by computing $Q^{(p)}$, a block-diagonal orthogonal matrix with 2^p orthogonal dense diagonal blocks that are computed in Line 3 of Algorithm 5.5.

The overall storage required to store Q equals to the sum of memory requirements for matrices $Q^{(i)}, 0 \leq i \leq p$. Assume that the off-diagonal ranks occurring in matrices $Q^{(i)}$, $0 \leq i < p$, are bounded by $\tilde{k}$. To determine the storage, we use that $Q^{(i)}$, for $0 \leq i < p$, has 2^i diagonal blocks of the form $[H_1\, H_2]$, where the storage of both H_1 and H_2 requires $\mathcal{O}(\tilde{k}\frac{n}{2^i} \log \frac{n}{2^i})$ memory. Hence we get that the storage for matrices $Q^{(i)}$, $0 \leq i < p$, adds

up to

$$\sum_{l=0}^{p-1} \tilde{k} 2^{l+1} \frac{n}{2^l} \log \frac{n}{2^l} = 2\tilde{k}n \sum_{l=0}^{p-1} \log \frac{n}{2^l} = 2\tilde{k}n \left(p \log n - \frac{(p-1)p}{2} \log 2 \right). \tag{5.12}$$

Moreover, the storage of a block diagonal matrix $Q^{(p)}$ with 2^p dense diagonal blocks requires $2^p n_{\text{stop}}^2 = n n_{\text{stop}}$ units of memory. Hence, from the latter and (5.12) follows that the overall memory needed for storing the matrix of eigenvectors Q is $\mathcal{O}(\tilde{k}n \log^2 n)$.

5.3.4 Computational complexity

Now we derive the theoretical complexity of Algorithm 5.5, based on the complexity of operations presented in Table 2.6. Additionally, the numerical results presented in Section 5.4 give an insight how the algorithm behaves in practice, and moreover, they confirm the theoretical results.

We first note that for a HODLR matrix of size $m \times m$ and with the off-diagonal rank k the complexity of one divide step, computed in Line 5–Line 9, is $\mathcal{O}(k^2 m \log^2 m)$. When performing level l divide step, the computation involves 2^l HODLR matrices of size $n/2^l \times n/2^l$. Denoting with $\tilde{k}$ the maximal off-diagonal rank appearing in the process, similarly as in the previous section we derive the computational complexity of our algorithm:

$$\begin{aligned} &\sum_{l=0}^{p-1} \tilde{k}^2 2^l \frac{n}{2^l} \log^2 \frac{n}{2^l} - \tilde{k}^2 n \sum_{l=0}^{p-1} \log^2 \frac{n}{2^l} \\ &\qquad = \tilde{k}^2 n p \left(\log^2 n - (p-1) \log n \log 2 + \frac{(p-1)(2p-1)}{6} \log^2 2 \right). \end{aligned} \tag{5.13}$$

At the final level of recursive application of Algorithm 5.5, when the algorithm is applied to matrices of size not larger than n_{stop}, the complexity comes from diagonalizing 2^p dense matrices, i.e., equals to $\mathcal{O}(n n_{\text{stop}}^2)$. Therefore, from (5.13) we obtain that the overall computational complexity of Algorithm 5.5 is $\mathcal{O}(\tilde{k}^2 n \log^3 n)$.

5.4 Numerical experiments

In this section, we show the performance of our MATLAB implementation of the spectral divide-and-conquer method in the HODLR format for various matrices. All computations were performed in MATLAB version R2016b on a machine with the dual Intel Core i7-5600U 2.60GHz CPU, 256 KByte of level 2 cache and 12 GByte of RAM, using a single core. The memory requirements shown in Example 5.12 are obtained experimentally, using MATLAB built-in functions.

In all experiments, we set the truncation tolerance to $\epsilon = 10^{-10}$, the minimal block-size $n_{\min} = 250$ for tridiagonal matrices and $n_{\min} = 500$ for b-banded matrices with $b > 1$. We use $n_b = 32$ as the minimal column block size in Algorithm 2.12. Moreover, the stopping tolerance in the hDWH algorithm is set to $\varepsilon = 10^{-15}$. In Algorithm 5.4 we use the oversampling parameter $s = 10$. We use breakeven points from Chapter 4 to set the termination criterion in Algorithm 5.5. For tridiagonal matrices we use $n_{\text{stop}} = 3250$, for 2-banded matrices $n_{\text{stop}} = 1750$ and $n_{\text{stop}} = 3000$ for b-banded with $b > 2$.

The efficiency of our algorithm is tested on a set of matrices coming from applications, as well as on various synthetic matrices.

5.4.1 Generation of test matrices

To generate synthetic matrices, we employ the procedure explained in Section 4.5.1 that uses a sequence of Givens rotations to obtain a symmetric banded matrix with a prescribed bandwidth and spectrum, starting from a diagonal matrix containing n eigenvalues. As the accuracy of computed spectral projectors depends on the relative spectral gap, we generate matrices such that gap is constant whenever the spectrum is split in half. We generate such a spectrum by first dividing the interval $[-1, 1]$ into $[-1, -\mathsf{gap}] \cup [\mathsf{gap}, 1]$ and then recursively apply the same procedure to both subintervals. In particular, interval $[c, d]$ is split into $[c, \frac{c+d}{2} - \frac{d-c}{2}\,\mathsf{gap}] \cup [\frac{c+d}{2} + \frac{d-c}{2}\,\mathsf{gap}, d]$. The recursive division stops when the number of subintervals is $\leq n/n_{\text{stop}}$. We assign equal number of eigenvalues coming from a uniform distribution to each subinterval on the lowest level of recursion. We note that similar results for eigenvalues coming from a geometric distribution have been observed, but we omit them to avoid redundancy.

5.4.2 Results for the hDWH algorithm

Example 5.8 (Performance relative to gap). The purpose of this example is to verify the accuracy of the newly proposed implementation of the QR-based iteration from Section 5.1. We investigate the behavior of the errors arising in the computation of spectral projectors $\Pi_{<0}$, which are also used in Section 4.5. In order to work with matrices with a prescribed gap, we consider $10\,000 \times 10\,000$ tridiagonal matrices with eigenvalues in $[-1, -\mathsf{gap}] \cup [\mathsf{gap}, 1]$, where gap varies from 10^{-15} to 10^{-1}. Moreover, only for the purpose of this example we consider tridiagonal matrices as general symmetric matrices in Algorithm 5.2. Figure 5.2 shows that, when incorporating the new QR decomposition for the first step of the hDWH alfgorithm, the errors exhibit similar behavior as in the hQDWH algorithm; see Figure 4.6 (left) for comparison. This confirms the efficiency of the new method.

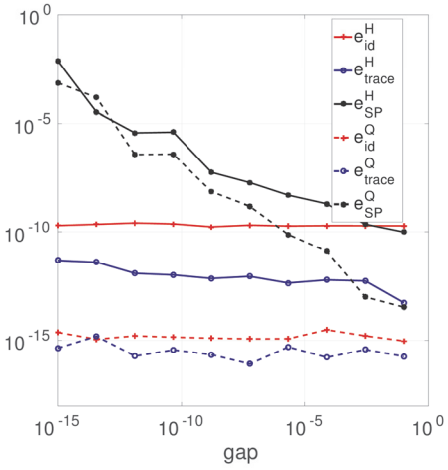

Figure 5.2 – Example 5.8. Comparison of the accuracy for the hDWH algorithm, with a new implementation of the first iterate, and the QDWH algorithm applied to matrices with varying spectral gap.

5.4.3 Results for the hSDC algorithm

Example 5.9 (Percentage and conditioning of selected columns). We first investigate the percentage of selected columns throughout Algorithm 5.5 depending on a given threshold δ, together with the condition number of the selected columns. We show results for matrices of size $n = 10240$, with bandwidths $b = 1$ and $b = 8$, and spectral gaps $\mathsf{gap} = 10^{-2}$ and $\mathsf{gap} = 10^{-6}$, generated as described above. In this example we ensure

that in all divide steps of Algorithm 5.5 the gap between separated parts of the spectrum corresponds to gap, by computing the shift μ as the median of eigenvalues of a considered matrix. In each divide step in Algorithm 5.5 we compute the percentage of selected columns, and finally we show their average for each δ. Moreover, we present a maximal condition number of the selected columns in the whole divide-and-conquer process for a given δ. As expected, smaller values of δ lead to a higher percentage of selected columns; however, the chosen columns exhibit a higher condition number as well. Figure 5.3 and Figure 5.4 show that already for $\delta \geq 0.4$ we get a good trade-off between the percentage of selected columns and their condition number. This also implies that the off-diagonal ranks in the eigenvectors matrix remain low.

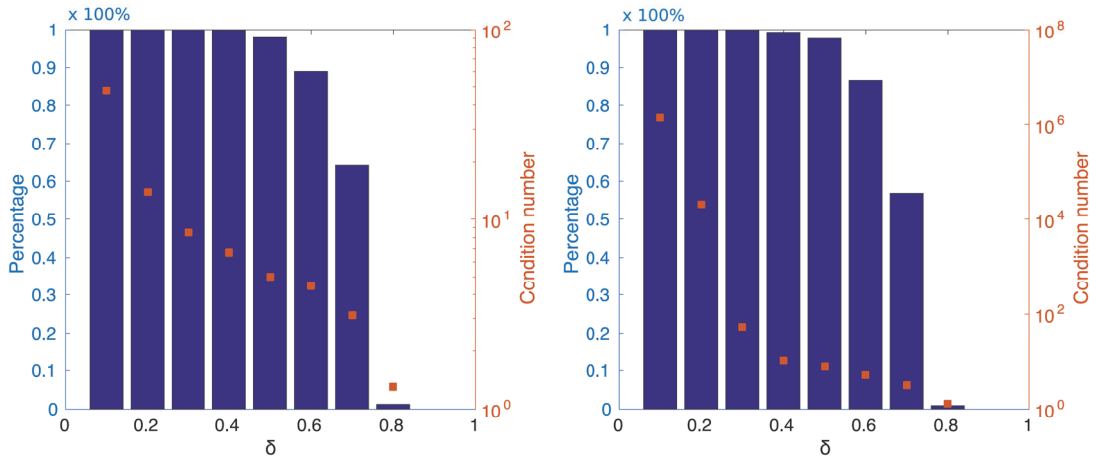

Figure 5.3 – Example 5.9. Percentage of selected columns and their condition number for a tridiagonal matrix with eigenvalues in $[-1, 1]$ *with relative spectral gap* $\mathsf{gap} = 10^{-2}$ *(left) and* $\mathsf{gap} = 10^{-6}$ *(right).*

Example 5.10 (Breakeven point relative to `eig`). Our runtime comparisons are performed on generated $n \times n$ banded matrices. We examine for which values of n Algorithm 5.5 outperforms `eig`. In Table 5.2 we show breakeven points for banded matrices constructed as in Section 5.4.1, with $\mathsf{gap} \in \{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$. We use the threshold parameter $\delta = 0.4$. For $b = 2$ and $b = 4$ banded matrices our algorithm becomes faster than `eig` for relatively small n. This is due to the fact that MATLAB's `eig` first performs tridiagonal reduction. Our results show the benefit of avoiding the reduction of a banded matrix to a tridiagonal form, especially when the involved bandwidth is small. Moreover, for tridiagonal matrices the breakeven point is attainable, but still relatively high.

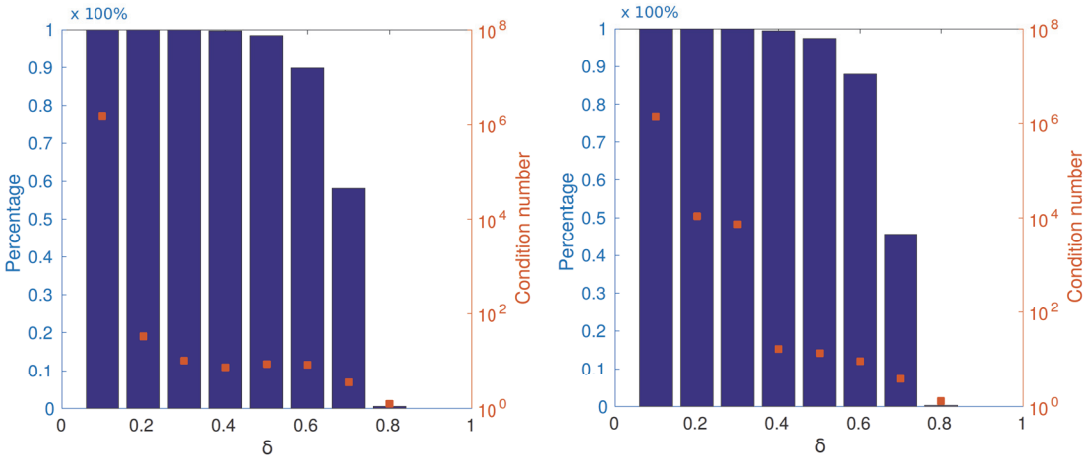

Figure 5.4 – Example 5.9. Percentage of selected columns and their condition number for a 8-banded matrix with eigenvalues in $[-1, 1]$ with relative spectral gap gap $= 10^{-2}$ *(left) and* gap $= 10^{-6}$ *(right).*

Table 5.2 – Breakeven point of hSDC relative to `eig` *applied for banded matrices with various bandwidths and spectral gaps.*

gap \ b	1	2	4	8
10^{-1}	$n = 11200$	$n = 2100$	$n = 3200$	$n = 5300$
10^{-2}	$n = 15600$	$n = 2500$	$n = 3800$	$n = 8200$
10^{-3}	$n = 17200$	$n = 2800$	$n = 4900$	$n = 8900$
10^{-4}	$n = 18500$	$n = 3000$	$n = 5200$	$n = 9500$

Example 5.11 (Accuracy for various matrices). In this example, we test the accuracy of the computed spectral decomposition. Denoting with $Q = [q_1, \ldots, q_n]$ and $\Lambda = \operatorname{diag}(\lambda_1, \ldots, \lambda_n)$ the output of Algorithm 5.5, and with $\tilde{Q} = [\tilde{q}_1, \ldots, \tilde{q}_n]$ and $\tilde{\Lambda} = \operatorname{diag}(\tilde{\lambda}_1, \ldots, \tilde{\lambda}_n)$ the eigenvalue decomposition obtained using MATLAB's `eig`, we consider four different error metrics:

- the largest relative error in the computed eigenvalues: $e_\lambda = \max_i |\lambda_i - \tilde{\lambda}_i| / \|A\|_2$,
- the largest relative residual norm: $e_{\text{res}} = \max_i \|Aq_i - \lambda_i q_i\|_2 / \|A\|_2$,
- the loss of orthogonality: $e_{\text{orth}} = \max_i \|Q^T q_i - e_i\|_2$,
- the largest error in the computed eigenvectors : $e_Q = \max_i |\sin \measuredangle(q_i, \tilde{q}_i)|$.

In the subsequent experiments we set $\delta = 0.4$.

1. We show the accuracy of the newly proposed algorithm for symmetric matrices arising from applications, as well as some of the matrices suggested in [86]. For matrices of size smaller than 3000, we use $n_{\text{stop}} = 500$, which allows us to perform at least one divide step in Algorithm 5.5.

   - the BCSSTRUC1 set in the Harwell-Boeing Collection [38]. Considered problems are in fact generalized eigenvalue problems, with M a mass matrix and K a stiffnes matrix. Each problem is transformed into an equivalent standard eigenvalue problem $L^{-1}KL^{-T}x = \lambda x$, where L denotes the Cholesky factor of M. Finally, matrices are reduced to tridiagonal form via MATLAB function `hess`.
   - The symmetric Alemdar and Cannizzo matrix [38], as well as several matrices from the NASA set [38]. These matrices are reduced to tridiagonal form using MATLAB function `hess`.
   - The $(1, 2, 1)$ symmetric tridiagonal Toeplitz matrix.
   - The Legendre-type tridiagonal matrix.
   - The Laguerre-type tridiagonal matrix.
   - The Hermite-type tridiagonal matrix.
   - Symmetric tridiagonal matrices with eigenvalues coming from a random $(0, 1)$ distribution.
   - Symmetric tridiagonal matrices with eigenvalues coming from a uniform distribution on $[-1, 1]$.

   In Table 5.3 we report the observed accuracies. The results are satisfactory, and the errors $e_\lambda, e_{\text{res}}$ and e_{orth} are roughly of order of the truncation tolerance $\epsilon = 10^{-10}$. However, the error e_Q varies from 1 to 10^{-8}. These results are a consequence of small distances between the eigenvalues, and the observed results can be theoretically justified by the $\sin\theta$ theorem [37]. In particular, we note that for the examples where the error e_Q is close to 1, i.e. for matrices bcsst08, bcsst09, bcstt11 and the Alemdar matrix, the smallest non-zero relative gaps between the eigenvalues are of order 10^{-19}, 10^{-18}, 10^{-17} and 10^{-17}, respectively.

Table 5.3 – Accuracy of the hSDC algorithm for tridiagonal matrices considered in Example 5.11.

	matrix	n	e_λ	e_{orth}	e_{res}	e_Q
BCSSTRUC1	bcsst08	1074	$2.2 \cdot 10^{-14}$	$1.3 \cdot 10^{-10}$	$1.4 \cdot 10^{-12}$	$7.1 \cdot 10^{-1}$
	bcsst09	1083	$4.1 \cdot 10^{-11}$	$4.9 \cdot 10^{-11}$	$6.3 \cdot 10^{-11}$	$9.9 \cdot 10^{-1}$
	bcsst11	1473	$2.8 \cdot 10^{-11}$	$3.3 \cdot 10^{-10}$	$1.7 \cdot 10^{-10}$	1
NASA	nasa1824	1824	$9.7 \cdot 10^{-13}$	$2.3 \cdot 10^{-10}$	$1.1 \cdot 10^{-11}$	$7.3 \cdot 10^{-7}$
	nasa2146	2146	$4.9 \cdot 10^{-12}$	$1.7 \cdot 10^{-10}$	$3.5 \cdot 10^{-11}$	$3.6 \cdot 10^{-8}$
	nasa2190	2190	$8.2 \cdot 10^{-13}$	$6.1 \cdot 10^{-10}$	$3.2 \cdot 10^{-12}$	$6.5 \cdot 10^{-6}$
	nasa4704	4704	$8.9 \cdot 10^{-12}$	$2.9 \cdot 10^{-10}$	$6.9 \cdot 10^{-12}$	$1.6 \cdot 10^{-5}$
	Cannizzo matrix	4098	$3.4 \cdot 10^{-11}$	$7.6 \cdot 10^{-10}$	$8.3 \cdot 10^{-10}$	$8.8 \cdot 10^{-4}$
	Alemdar matrix	6245	$1.6 \cdot 10^{-11}$	$9.2 \cdot 10^{-10}$	$2.4 \cdot 10^{-11}$	1
	(1,2,1) matrix	10000	$1.3 \cdot 10^{-11}$	$8.6 \cdot 10^{-11}$	$7.3 \cdot 10^{-10}$	$4.4 \cdot 10^{-7}$
	Clement-type	10000	$1.5 \cdot 10^{-11}$	$3.8 \cdot 10^{-10}$	$2.5 \cdot 10^{-11}$	$1.3 \cdot 10^{-7}$
	Legendre-type	10000	$2.7 \cdot 10^{-11}$	$8.4 \cdot 10^{-11}$	$1.4 \cdot 10^{-11}$	$2.8 \cdot 10^{-7}$
	Laguerre-type	10000	$3.8 \cdot 10^{-11}$	$7.8 \cdot 10^{-10}$	$3.8 \cdot 10^{-10}$	$2.5 \cdot 10^{-7}$
	Hermite-type	10000	$3 \cdot 10^{-11}$	$3.9 \cdot 10^{-10}$	$1.7 \cdot 10^{-10}$	$6.6 \cdot 10^{-8}$
	Random normal $(0, 1)$	10000	$1.4 \cdot 10^{-11}$	$1.1 \cdot 10^{-10}$	$3.9 \cdot 10^{-10}$	$8.4 \cdot 10^{-6}$
	Uniform in $[-1, 1]$	10000	$5.1 \cdot 10^{-11}$	$6.5 \cdot 10^{-10}$	$3.1 \cdot 10^{-10}$	$4.8 \cdot 10^{-6}$

We also mention that the percentage of the selected columns, as well as the condition number of the selected columns throughout Algorithm 5.5 were along the lines the results presented in Example 5.9.

2. The aim of this part is to examine the dependency of the error measures defined above on a decreasing spectral gap. To this end, we construct symmetric 8-banded matrices of size $n = 10240$ with $\mathsf{gap} = 10^{-i}, i = 1, \ldots, 10$ using a method from Section 5.4.1. As in Example 5.9, we ensure that the gap between two separated parts of spectrum in all divide steps of Algorithm 5.5 is the prescribed gap. Figure 5.5 shows that our algorithm preforms well for matrices with larger spectral gaps, and confirms the expected behavior of errors, given that the used truncation tolerance is $\epsilon = 10^{-10}$. The error growth is a result of a decreasing accuracy when computing spectral projectors associated with decreasing spectral gaps. Similarly as in the previous part, the error e_Q exhibits growth from 10^{-8} to 1 also as a result of decreasing spectral gaps.

Example 5.12 (Scalability). For $n \times n$ tridiagonal matrices, generated as in Section 5.4.1

with $\mathsf{gap} = 10^{-2}$, we demonstrate the performance of our algorithm with respect to to the matrix size n. Again we use the threshold parameter $\delta = 0.4$. We show that the asymptotic behavior of our algorithm matches the theoretical bounds both for the computational time and storage requirements. Figure 5.6 (left) shows that time needed to compute the complete spectral decomposition follows the expected $\mathcal{O}(n \log^3(n))$ reference line, whereas Figure 5.6 (right) demonstrates that the memory required to store the matrix of eigenvectors has $\mathcal{O}(n \log^2(n))$ behavior.

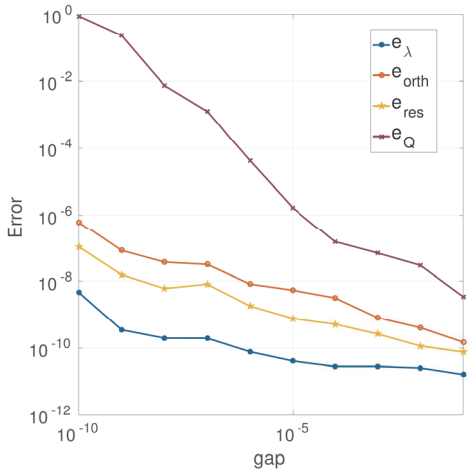

Figure 5.5 – Example 5.11 (2). Behavior of the errors in the hSDC algorithm with respect to a decreasing spectral gap for symmetric 8-banded matrices of size $n = 10240$.

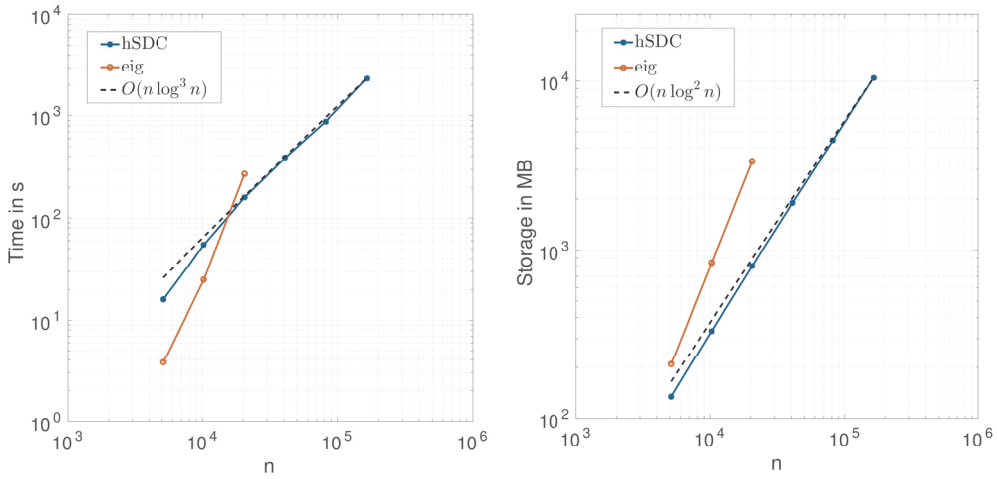

Figure 5.6 – Example 5.12. Performance of the hSDC algorithm and `eig` *with respect to n for tridiagonal matrices. Left: Computational time. Right: Memory requirements.*

5.5 Conclusion

In this work we have proposed a new fast spectral divide-and-conquer algorithm for computing the complete spectral decomposition of symmetric banded matrices. The algorithm exploits the fact that spectral projectors of banded matrices can be efficiently computed and stored in the HODLR format. We have presented a fast novel method for selecting well-conditioned columns of a spectral projector based on a Cholesky decomposition with pivoting, and provided a theoretical justification for the method. This method enables us to efficiently split the computation of the spectral decomposition of a symmetric HODLR matrix into two smaller subproblems.

The new spectral D&C method is implemented in the HODLR format and has a linear-polylogarithmic complexity. In the numerical experiments, performed both on synthetic matrices and matrices coming from applications, we have verified the efficiency of our method, and have shown that it is a competitive alternative to the state-of-the-art methods for some classes of banded matrices.

6 Conclusion

In this thesis we have focused on the development of fast algorithms for addressing and solving important problems in the numerical linear algebra field: QR decomposition, spectral projectors and the symmetric eigenvalue problem. Our main tool is a subset of hierarchical matrices, the HODLR format, that requires almost linear storage cost, and allows for standard operations, such as matrix-vector multiplication, matrix-matrix addition and multiplication, and matrix factorizations, to be performed in the approximative arithmetic with linear-polylogarithmic complexity.

After reviewing the HODLR arithmetic, in Chapter 2 we have developed a new method for computing QR decompositions of HODLR matrices. As mentioned above, several standard matrix operations already have efficient implementations in the HODLR format. However, three different algorithms that exist in the literature for computing a QR decomposition of a hierarchical matrix seem to have certain drawbacks. Indeed, alongside favorable complexity, the QR decomposition ought to be accurate and the orthogonality in the factor Q attained. The algorithms developed so far show that for hierarchical matrices it is rather difficult to meet all the requirements. We have derived a new method based on a dense block QR decomposition that seems to have all desirable properties. Our method is performed in a recursive block column manner, with the orthogonal factor stored in terms of a compact WY representation in the HODLR format. The overall cost of the method is $\mathcal{O}(k^2 n \log^2 n)$, where k is the off-diagonal rank, and n dimension of a HODLR matrix. Moreover, the numerical results indicate that the method yields an accurate decomposition, and preserves the orthogonality in Q, even for badly conditioned matrices.

In Chapter 4 we have derived a fast method for the approximate computation of spectral projectors for symmetric banded matrices. Additionally, we have focused on the case when the associated spectral gap is small, which poses problems in linear scaling approaches based on approximate sparsity. The proposed method is built upon a variant of the QR-based dynamically weighted Halley (QDWH) iteration [89, 91], which computes the polar decomposition. In our approach, iterates of the QDWH algorithm are implemented in the HODLR format, such that the overall algorithm has linear-polylogarithmic complexity. The first iteration of the QDWH algorithm has been discussed in detail. Moreover, we have derived theoretical justifications and a fast novel method for representing the first iterate exactly in the HODLR format.

One major theoretical contribution is an a priori bound for the singular values in the off-diagonal blocks of spectral projectors. This theoretical result serves as a tool to show that memory needed to store approximate spectral projectors in the HODLR format depends only logarithmically on the spectral gap. In fact, this presents a big improvement in comparison to approximate sparsity, where the spectral gap enters memory requirements inverse proportionally. Numerical results show that our MATLAB implementation becomes faster than `eig` already for matrix sizes of a few thousand. The effectiveness of our method is demonstrated for small spectral gaps, and matrices of large size. Additionally, the results provide evidence that the theoretical asymptotic $\mathcal{O}(k^2 n \log^2 n)$ complexity is attained.

In Chapter 5 we have presented a new method for computing a complete spectral decomposition of a banded matrix in the HODLR format. Our approach follows the spectral divide-and-conquer method derived in [91], which computes a complete set of eigenvalues and eigenvectors by recursively dividing a larger eigenvalue problem into smaller ones by splitting the spectrum. In particular, this is performed by extracting invariant subspaces associated with a part of the spectrum from related spectral projectors, where the computation of spectral projectors is carried out in the HODLR format as aforementioned. In order to determine an invariant subspace from a spectral projector, one needs to find a basis for the range of the spectral projector. Given that spectral projectors are rank deficient and represented in the HODLR format, this presents a great challenge. A major contribution in this chapter is a novel fast method for finding a basis for the range of a spectral projector, and its theoretical analysis.

Analogously to Chapter 4, we have addressed the computation of the first iterate in

the QDWH algorithm for general symmetric HODLR matrices. We have extended the method derived in Chapter 2 to obtain an accurate representation of the first iterate. This allows for our spectral divide-and-conquer method to be successfully employed for matrices with small spectral gaps. The matrix of eigenvectors is returned in a factored form, given in terms of orthonormal factors, where each factor is an orthonormal basis for an invariant subspace. The memory required to represent the matrix of eigenvectors is $\mathcal{O}(kn \log^2 n)$. Numerical results demonstrate that the method exhibits $\mathcal{O}(k^2 n \log^3 n)$ computational complexity for obtaining a complete spectral decomposition. Moreover, for several synthetic examples, and for matrices from various applications, the results indicate that the method yields accurate spectral decompositions.

Bibliography

[1] N. I. Akhiezer. *Elements of the Theory of Elliptic Functions*, volume 79 of *Translations of Mathematical Monographs*. American Mathematical Society, Providence, RI, 1990.

[2] S. Ambikasaran, D. Foreman-Mackey, L. Greengard, D. W. Hogg, and M. O'Neil. Fast direct methods for gaussian processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(2):252–265, 2016.

[3] A. H. Aminfar and E. Darve. A fast and memory efficient sparse solver with applications to finite-element matrices. arXiv:1410.2697, 2014.

[4] P. Arbenz. Divide and conquer algorithms for the bandsymmetric eigenvalue problem. *Parallel Comput.*, 18(10):1105–1128, 1992.

[5] T. Auckenthaler, V. Blum, H.-J. Bungartz, T. Huckle, R. Johanni, L. Krämer, B. Lang, H. Lederer, and P. R. Willems. Parallel solution of partial symmetric eigenvalue problems from electronic structure calculations. *Parallel Computing*, 37(12):783–794, 2011.

[6] T. Auckenthaler, H.-J. Bungartz, T. Huckle, L. Krämer, B. Lang, and P. Willems. Developing algorithms and software for the parallel solution of the symmetric eigenvalue problem. *Journal of Computational Science*, 2(3):272–278, 2011.

[7] J. Ballani and D. Kressner. Sparse inverse covariance estimation with hierarchical matrices. Technical Report, EPFL-MATHICSE-ANCHP, 2014.

[8] J. Ballani and D. Kressner. *Matrices with Hierarchical Low-rank Structures*. CIME Summer School on Exploiting Hidden Structure in Matrix Computations, 2016.

[9] U. Baur and P. Benner. Factorized solution of Lyapunov equations based on hierarchical matrix arithmetic. *Computing*, 78(3):211–234, 2006.

[10] M. Bebendorf. Approximation of boundary element matrices. *Numer. Math.*, 86(4):565–589, 2000.

[11] M. Bebendorf. *Hierarchical Matrices*, volume 63 of *Lecture Notes in Computational Science and Engineering*. Springer-Verlag, Berlin, 2008.

[12] M. Bebendorf and W. Hackbusch. Existence of $\mathcal{H}$-matrix approximants to the inverse FE-matrix of elliptic operators with L^∞-coefficients. *Numer. Math.*, 95(1):1–28, 2003.

[13] M. Bebendorf and W. Hackbusch. Stabilized rounded addition of hierarchical matrices. *Numer. Linear Algebra Appl.*, 14(5):407–423, 2007.

[14] M. Bebendorf and S. Rjasanow. Adaptive low-rank approximation of collocation matrices. *Computing*, 70(1):1–24, 2003.

[15] B. Beckermann and A. Townsend. On the singular values of matrices with displacement structure. *SIAM J. Matrix Anal. Appl.*, 38(4):1227–1248, 2017.

[16] P. Benner, S. Börm, T. Mach, and K. Reimer. Computing the eigenvalues of symmetric $\mathcal{H}^2$-matrices by slicing the spectrum. *Comput. Vis. Sci.*, 16(6):271–282, 2013.

[17] P. Benner and T. Mach. On the QR decomposition of $\mathcal{H}$-matrices. *Computing*, 88(3-4):111–129, 2010.

[18] P. Benner and T. Mach. Computing all or some eigenvalues of symmetric $\mathcal{H}_\ell$-matrices. *SIAM J. Sci. Comput.*, 34(1):A485–A496, 2012.

[19] P. Benner and T. Mach. The LR Cholesky algorithm for symmetric hierarchical matrices. *Linear Algebra Appl.*, 439(4):1150–1166, 2013.

[20] P. Benner and T. Mach. The preconditioned inverse iteration for hierarchical matrices. *Numer. Linear Algebra Appl.*, 20(1):150–166, 2013.

[21] M. Benzi, P. Boito, and N. Razouk. Decay properties of spectral projectors with applications to electronic structure. *SIAM Rev.*, 55(1):3–64, 2013.

[22] M. Benzi and G. H. Golub. Bounds for the entries of matrix functions with applications to preconditioning. *BIT*, 39(3):417–438, 1999.

[23] M. Benzi and N. Razouk. Decay bounds and $O(n)$ algorithms for approximating functions of sparse matrices. *Electron. Trans. Numer. Anal.*, 28:16–39, 2007/08.

[24] M. Berljafa and S. Güttel. A rational krylov toolbox for Matlab. MIMS EPrint 2014.56, The University of Manchester, 2014.

[25] G. Beylkin, N. Coult, and M. J. Mohlenkamp. Fast spectral projection algorithms for density-matrix computations. *J. Comput. Phys.*, 152(1):32–54, 1999.

[26] P. Bientinesi, F. D Igual, D. Kressner, M. Petschow, and E. S. Quintana-Ortí. Condensed forms for the symmetric eigenvalue problem on multi-threaded architectures. *Concurrency and Computation: Practice and Experience*, 23(7):694–707, 2011.

[27] C. H. Bischof, B. Lang, and X. Sun. A framework for symmetric band reduction. *ACM Trans. Math. Software*, 26(4):581–601, 2000.

[28] Å. Björck. *Numerical Methods for Least Squares Problems*. SIAM, Philadelphia, PA, 1996.

[29] S. Börm. *Efficient Numerical Methods for Non-local Operators*, volume 14 of *EMS Tracts in Mathematics*. European Mathematical Society (EMS), Zürich, 2010. $\mathcal{H}^2$–Matrix Compression, Algorithms and Analysis.

[30] S. Börm and L. Grasedyck. Hybrid cross approximation of integral operators. *Numer. Math.*, 101(2):221–249, 2005.

[31] D. Braess and W. Hackbusch. Approximation of $1/x$ by exponential sums in $[1, \infty)$. *IMA J. Numer. Anal.*, 25(4):685–697, 2005.

[32] S. Chandrasekaran, M. Gu, and W. Lyons. A fast adaptive solver for hierarchically semiseparable representations. *Calcolo*, 42(3-4):171–185, 2005.

[33] S. Chandrasekaran, M. Gu, and T. Pals. A fast ULV decomposition solver for hierarchically semiseparable representations. *SIAM J. Matrix Anal. Appl.*, 28(3):603–622, 2006.

[34] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, third edition, 2009.

[35] E. Corona, P. G. Martinsson, and D. Zorin. An $O(N)$ direct solver for integral equations on the plane. *Appl. Comput. Harmon. Anal.*, 38(2):284–317, 2015.

[36] J. J. M. Cuppen. A divide and conquer method for the symmetric tridiagonal eigenproblem. *Numer. Math.*, 36(2):177–195, 1980/81.

[37] C. Davis and W. M. Kahan. The rotation of eigenvectors by a perturbation. III. *SIAM J. Numer. Anal.*, 7:1–46, 1970.

[38] T. A. Davis and Y. Hu. The University of Florida Sparse Matrix Collection. *ACM Trans. Math. Software*, 38(1):1–25, 2011.

[39] S. Delvaux and M. Van Barel. A Hessenberg reduction algorithm for rank structured matrices. *SIAM J. Matrix Anal. Appl.*, 29(3):895–926, 2007.

[40] S. Delvaux and M. Van Barel. Eigenvalue computation for unitary rank structured matrices. *J. Comput. Appl. Math.*, 213(1):268–287, 2008.

[41] S. Demko, W. F. Moss, and P. W. Smith. Decay rates for inverses of band matrices. *Math. Comp.*, 43(168):491–499, 1984.

[42] J. W. Demmel, O. A. Marques, B. N. Parlett, and C. Vömel. Performance and accuracy of LAPACK's symmetric tridiagonal eigensolvers. *SIAM J. Sci. Comput.*, 30(3):1508–1526, 2008.

[43] I. S. Dhillon. *A New $O(n^2)$ Algorithm for the Symmetric Tridiagonal Eigenvalue/Eigenvector Problem.* Doctoral thesis, University of California, Berkeley, 1997.

[44] I. S. Dhillon, B. N. Parlett, and C. Vömel. The design and implementation of the MRRR algorithm. *ACM Trans. Math. Software*, 32(4):533–560, 2006.

[45] Y. Eidelman and I. Gohberg. On generators of quasiseparable finite block matrices. *Calcolo*, 42(3-4):187–214, 2005.

[46] Y. Eidelman, I. Gohberg, and L. Gemignani. On the fast reduction of a quasiseparable matrix to Hessenberg and tridiagonal forms. *Linear Algebra Appl.*, 420(1):86–101, 2007.

[47] Y. Eidelman, I. Gohberg, and I. Haimovici. *Separable Type Representations of Matrices and Fast Algorithms. Vol. 1*, volume 234 of *Operator Theory: Advances*

and Applications. Birkhäuser/Springer, Basel, 2014. Basics. Completion problems. Multiplication and inversion algorithms.

[48] Y. Eidelman, I. Gohberg, and I. Haimovici. *Separable Type Representations of Matrices and Fast Algorithms. Vol. 2*, volume 235 of *Operator Theory: Advances and Applications*. Birkhäuser/Springer Basel AG, Basel, 2014. Eigenvalue method.

[49] L. Eldén. Algorithms for the regularization of ill-conditioned least squares problems. *Nordisk Tidskr. Informationsbehandling (BIT)*, 17(2):134–145, 1977.

[50] L. Eldén. An algorithm for the regularization of ill-conditioned, banded least squares problems. *SIAM J. Sci. Statist. Comput.*, 5(1):237–254, 1984.

[51] E. Elmroth and F. G. Gustavson. Applying recursion to serial and parallel qr factorization leads to better performance. *IBM Journal of Research and Development*, 44(4):605–624, 2000.

[52] J. G. F. Francis. The QR transformation: a unitary analogue to the LR transformation. I. *Comput. J.*, 4:265–271, 1961/1962.

[53] J. G. F. Francis. The QR transformation. II. *Comput. J.*, 4:332–345, 1961/1962.

[54] P. Gatto and J. S. Hesthaven. A preconditioner based on low-rank approximation of schur complements. arXiv:1508.07798, 2015.

[55] I. P. Gavrilyuk, W. Hackbusch, and B. N. Khoromskij. $\mathcal{H}$-matrix approximation for the operator exponential with applications. *Numer. Math.*, 92(1):83–111, 2002.

[56] I. P. Gavrilyuk, W. Hackbusch, and B. N. Khoromskij. Data-sparse approximation to the operator-valued functions of elliptic operator. *Math. Comp.*, 73(247):1297–1324, 2004.

[57] P. Ghysels, X. S. Li, F. H. Rouet, S. Williams, and A. Napov. An efficient multicore implementation of a novel HSS-structured multifrontal solver using randomized sampling. *SIAM J. Sci. Comput.*, 38(5):S358–S384, 2016.

[58] A. Gillman, P. M. Young, and P. G. Martinsson. A direct solver with $O(N)$ complexity for integral equations on one-dimensional domains. *Front. Math. China*, 7(2):217–247, 2012.

[59] S. Goedecker. Linear scaling electronic structure methods. *Rev. Mod. Phys.*, 71:1085–1123, 1999.

[60] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, MD, fourth edition, 2013.

[61] J. Gördes. *Eigenwertproblem von Hierarchischen Matrizen mit Lokalem Rang* 1. Diploma thesis, Universität zu Kiel, 2009.

[62] L. Grasedyck, W. Hackbusch, and B. N. Khoromskij. Solution of large scale algebraic matrix Riccati equations by use of hierarchical matrices. *Computing*, 70(2):121–165, 2003.

[63] L. Greengard, D. Gueyffier, P. G. Martinsson, and V. Rokhlin. Fast direct solvers for integral equations in complex three-dimensional domains. *Acta Numer.*, 18:243–275, 2009.

[64] M. Gu and S. C. Eisenstat. A divide-and-conquer algorithm for the symmetric tridiagonal eigenproblem. *SIAM J. Matrix Anal. Appl.*, 16(1):172–191, 1995.

[65] M. Gu and S. C. Eisenstat. Efficient algorithms for computing a strong rank-revealing QR factorization. *SIAM J. Sci. Comput.*, 17(4):848–869, 1996.

[66] W. Hackbusch. A sparse matrix arithmetic based on $\mathcal{H}$-matrices. I. Introduction to $\mathcal{H}$-matrices. *Computing*, 62(2):89–108, 1999.

[67] W. Hackbusch. *Hierarchical Matrices: Algorithms and Analysis*, volume 49 of *Springer Series in Computational Mathematics*. Springer, Heidelberg, 2015.

[68] W. Hackbusch and B. N. Khoromskij. A sparse $\mathcal{H}$-matrix arithmetic. II. Application to multi-dimensional problems. *Computing*, 64(1):21 47, 2000.

[69] W. Hackbusch, B. N. Khoromskij, and R. Kriemann. Hierarchical matrices based on a weak admissibility criterion. *Computing*, 73(3):207–243, 2004.

[70] W. Hackbusch and W. Kress. A projection method for the computation of inner eigenvalues using high degree rational operators. *Computing*, 81(4):259–268, 2007.

[71] A. Haidar, H. Ltaief, and J. Dongarra. Parallel Reduction to Condensed Forms for Symmetric Eigenvalue Problems Using Aggregated Fine-grained and Memory-aware

Kernels. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 8:1–8:11. ACM, 2011.

[72] A. Haidar, H. Ltaief, and J. Dongarra. Toward a high performance tile divide and conquer algorithm for the dense symmetric eigenvalue problem. *SIAM J. Sci. Comput.*, 34(6):C249–C274, 2012.

[73] A. Haidar, R. Solcà, M. Gates, S. Tomov, T. Schulthess, and J. Dongarra. Leading Edge Hybrid Multi-GPU Algorithms for Generalized Eigenproblems in Electronic Structure Calculations. In *Supercomputing*, volume 7905 of *Lecture Notes in Computer Science*, pages 67–80. Springer Berlin Heidelberg, 2013.

[74] N. Halko, P. G. Martinsson, and J. A. Tropp. Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Rev.*, 53(2):217–288, 2011.

[75] N. J. Higham. Computing the polar decomposition—with applications. *SIAM J. Sci. Statist. Comput.*, 7(4):1160–1174, 1986.

[76] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, Philadelphia, PA, 1996.

[77] N. J. Higham. *Functions of Matrices*. SIAM, Philadelphia, PA, 2008. Theory and computation.

[78] N. J. Higham and F. Tisseur. A block algorithm for matrix 1-norm estimation, with an application to 1-norm pseudospectra. *SIAM J. Matrix Anal. Appl.*, 21(4):1185–1201 (electronic), 2000.

[79] R. A. Horn and C. R. Johnson. *Topics in Matrix Analysis*. Cambridge University Press, Cambridge, 1991.

[80] D. Kressner, S. Massei, and L. Robol. Low-rank updates and a divide-and-conquer method for linear matrix equations. arXiv:1712.04349, 2017.

[81] D. Kressner and A. Šušnjara. Fast computation of spectral projectors of banded batrices. *SIAM J. Matrix Anal. Appl.*, 38(3):984–1009, 2017.

[82] V. N. Kublanovskaya. Some algorithms for the solution of the complete problem of eigenvalues. *Ž. Vyčisl. Mat. i Mat. Fiz.*, 1:555–570, 1961.

[83] M. Lintner. *Lösung der 2D Wellengleichung mittels hierarchischer Matrizen*. Doctoral thesis, TU München, 2002.

[84] M. Lintner. The eigenvalue problem for the 2D Laplacian in $\mathcal{H}$-matrix arithmetic and application to the heat and wave equation. *Computing*, 72(3-4):293–323, 2004.

[85] T. Mach. *Eigenvalue Algorithms for Symmetric Hierarchical Matrices*. Doctoral thesis, TU Chemnitz, 2012.

[86] O. A. Marques, C. Vömel, J. W. Demmel, and B. N. Parlett. Algorithm 880: a testing infrastructure for symmetric tridiagonal eigensolvers. *ACM Trans. Math. Software*, 35(1):Art. 8, 13, 2009.

[87] P. G. Martinsson and V. Rokhlin. A fast direct solver for boundary integral equations in two dimensions. *J. Comput. Phys.*, 205(1):1–23, 2005.

[88] C. B. Moler and G. W. Stewart. On the Householder-Fox algorithm for decomposing a projection. *J. Comput. Phys.*, 28(1):82–91, 1978.

[89] Y. Nakatsukasa, Z. Bai, and F. Gygi. Optimizing Halley's iteration for computing the matrix polar decomposition. *SIAM J. Matrix Anal. Appl.*, 31(5):2700–2720, 2010.

[90] Y. Nakatsukasa and R. W. Freund. Computing fundamental matrix decompositions accurately via the matrix sign function in two iterations: The power of Zolotarev's functions. *SIAM Rev.*, 2016.

[91] Y. Nakatsukasa and N. J. Higham. Stable and efficient spectral divide and conquer algorithms for the symmetric eigenvalue decomposition and the SVD. *SIAM J. Sci. Comput.*, 35(3):A1325–A1349, 2013.

[92] A. Napov and X. S. Li. An algebraic multifrontal preconditioner that exploits the low-rank property. *Numer. Linear Algebra Appl.*, 23(1):61–82, 2016. nla.2006.

[93] J. M. Ostrowski, M. Bebendorf, R. Hiptmair, and F. Kramer. $\mathcal{H}$-matrix-based operator preconditioning for full Maxwell at low frequencies. *IEEE Transactions on Magnetics*, 46(8):3193–3196, Aug 2010.

[94] B. N. Parlett. *The Symmetric Eigenvalue Problem*. SIAM, Philadelphia, PA, 1998.

[95] P. P. Petrushev and V. A. Popov. *Rational Approximation of Real Functions*, volume 28 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, Cambridge, 1987.

[96] M. Petschow, E. Peise, and P. Bientinesi. High-performance solvers for dense Hermitian eigenproblems. *SIAM J. Sci. Comput.*, 35(1):C1–C22, 2013.

[97] H. Pouransari, P. Coulier, and E. Darve. Fast hierarchical solvers for sparse matrices using extended sparsification and low-rank approximation. arXiv:1510.07363, 2015.

[98] H. Rutishauser. Solution of eigenvalue problems with the LR-transformation. *Nat. Bur. Standards Appl. Math. Ser.*, 1958(49):47–81, 1958.

[99] Y. Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, Philadelphia, PA, second edition, 2003.

[100] Y. Saad. *Numerical Methods for Large Eigenvalue Problems*, volume 66 of *Classics in Applied Mathematics*. SIAM, Philadelphia, PA, 2011. Revised edition of the 1992 original [1177405].

[101] R. Schreiber and C. Van Loan. A storage-efficient WY representation for products of Householder transformations. *SIAM J. Sci. Statist. Comput.*, 10(1):53–57, 1989.

[102] H. R. Schwarz. Handbook Series Linear Algebra: Tridiagonalization of a symmetric band matrix. *Numer. Math.*, 12(4):231–241, 1968.

[103] S. Si, C. J. Hsieh, and I. S. Dhillon. Memory efficient kernel approximation. *Journal of Machine Learning Research*, 18(20):1–32, 2017.

[104] H. D. Simon and H. Zha. Low-rank matrix approximation using the Lanczos bidiagonalization process with applications. *SIAM J. Sci. Comput.*, 21(6):2257–2274, 2000.

[105] E. Solomonik, G. Ballard, J. Demmel, and T. Hoefler. A communication-avoiding parallel algorithm for the symmetric eigenvalue problem. arXiv:1604.03703, 2016.

[106] A. Stathopoulos and K. Wu. A block orthogonalization procedure with constant synchronization requirements. *SIAM J. Sci. Comput.*, 23(6):2165–2182, 2002.

[107] G. W. Stewart. *Matrix Algorithms. Vol. I.* SIAM, Philadelphia, PA, 1998. Basic decompositions.

[108] A. Šušnjara and D. Kressner. A fast spectral divide-and-conquer method for banded matrices. arXiv:1801.04175 [math.NA], 2018.

[109] R. Vandebril, M. Van Barel, and N. Mastronardi. *Matrix Computations and Semiseparable Matrices. Vol. 1.* Johns Hopkins University Press, Baltimore, MD, 2008.

[110] R. Vandebril, M. Van Barel, and N. Mastronardi. *Matrix Computations and Semiseparable Matrices. Vol. II.* Johns Hopkins University Press, Baltimore, MD, 2008. Eigenvalue and Singular Value Methods.

[111] J. Vogel, J. Xia, S. Cauley, and V. Balakrishnan. Superfast divide-and-conquer method and perturbation analysis for structured eigenvalue solutions. *SIAM J. Sci. Comput.*, 38(3):A1358–A1382, 2016.

[112] R. Wang, R. Li, M.W. Mahoney, and E. Darve. Structured block basis factorization for scalable kernel matrix evaluation. arXiv:1505.00398, 2015.

[113] S. Wang, X. S. Li, F. H. Rouet, J. Xia, and M. V. de Hoop. A parallel geometric multifrontal solver using hierarchically semiseparable structure. *ACM Trans. Math. Software*, 42(3):Art. 21, 21, 2016.

[114] Y. Xi, J. Xia, and R. Chan. A fast randomized eigensolver with structured LDL factorization update. *SIAM J. Matrix Anal. Appl.*, 35(3):974–996, 2014.

[115] J. Xia, S. Chandrasekaran, M. Gu, and X. S. Li. Fast algorithms for hierarchically semiseparable matrices. *Numer. Linear Algebra Appl.*, 17(6):953–976, 2010.

[116] Y. Yamamoto, Y. Nakatsukasa, Y. Yanagisawa, and T. Fukaya. Roundoff error analysis of the CholeskyQR2 algorithm. *Electron. Trans. Numer. Anal.*, 44:306–326, 2015.

Curriculum Vitae

Personal information

Name	Ana Šušnjara
Date of birth	09.09.1989.
Place of birth	Split, Croatia
Nationality	Croatian

Education

since Oct 2013	**Doctoral studies in Applied Mathematics** École polytechnique fédérale de Lausanne Thesis: *Fast hierarchical solvers for symmetric eigenvalue problems* Phd Advisor: Prof. D. Kressner
2011 – 2013	**M.Sc. in Applied Mathematics** Department of Mathematics, Faculty of Science, University of Zagreb Thesis: *Axially symmetric model of the mitral valve.* Summa cum Laude. Master Thesis advisor: Prof. J. Tambača
2008 – 2011	**B.Sc. in Mathematics** Department of Mathematics, Faculty of Science, University of Zagreb
2004–2008	**V. Gymnasium**, Zagreb, Croatia

Work experience

since Oct 2013	**Research and teaching assistant** École polytechnique fédérale de Lausanne
2009–2013	**Student teaching assistant** Department of Mathematics, Faculty of Science, University of Zagreb

Volunteering experience

2016 2017	EPFL SIAM Student Chapter Vice President
2015– 2016	EPFL SIAM Student Chapter Secretary
2008–2012	Math Competitions Tutor for high-school students (preparation for national and international competitions)

Awards and scholarships

Sep 2017	Award for exceptional teaching contributions in the academic year 2016/2017
Jun 2014	Best poster award at IWASEP10
Jun 2013	Award of the Program Directors' Council, Department of Mathematics for outstanding achievements during the M.Sc. studies
Apr 2013	Award of the Faculty Council, Faculty of Science for the best student in M.Sc. in Applied Mathematics
Jun 2012	University of Zagreb Chancellor's Award
Jul 2011	Award of the Program Directors' Council, Department of Mathematics for outstanding achievements in the academic year 2010/11
2010 – 2013	City of Zagreb Scholarship
2008 – 2010	Croatian Ministry of Education, Science and Sport Scholarship

Publications and preprints

- A. Šušnjara and D. Kressner. *A fast spectral divide-and-conquer method for banded matrices.* arXiv:1801.04175 [math.NA], 2018.
- D. Kressner and A. Šušnjara. *Fast computation of spectral projectors of banded matrices. SIAM J. Matrix Anal. Appl.*, 38(3):984–1009, 2017.
- A. Šušnjara, N. Perraudin, D. Kressner, and P. Vandergheynst. *Accelerated filtering on graphs using Lanczos method.* arXiv:1509.04537, 2015.
- M. Ornik and A. Šušnjara. *Neki prilozi teoriji egzaktne rekonstrukcije poligona (Some contributions to the theory of exact polygon reconstruction).* University of Zagreb Chancellor's Award Winning Research Paper. Supervised by Prof. Z. Drmač.
- M. Berljafa and A. Šušnjara. *Regularni politopi (Regular polytopes)*, Croatian Mathematical Electronic Journal math.e Vol. 18

Conferences and schools

- NumPDE Summer Retreat, Aug 9–11 2017, Disentis, Switzerland. Talk: *Fast computation of spectral projectors of banded matrices.*
- Householder Symposium XX, June 19–23 2017, Blacksburg, USA. Poster: *Fast computation of spectral projectors of banded matrices.*
- MATHICSE retreat, June 14–16 2017, Leysin, Switzerland. Talk: *Fast algorithms for computation of spectral projectors and spectral decompositions.*
- SIAM CSE17, Feb 27–Mar 3 2017, Atlanta, USA. Talk: *Fast computation of spectral projectors of banded matrices.*
- MATHICSE retreat, July 13–15 2015, Leysin, Switzerland. Talk: *On low-rank updates of matrix functions.*
- Gene Golub SIAM Summer School, *RandNLA: Randomization in Numerical Linear Algebra*, June 15–26 2015, Delphi, Greece.
- Swiss Numerical Analysis Day, April 17 2015, University of Geneva, Switzerland. Poster: *On low-rank updates of matrix functions.*
- Student Krylov Day, February 2 2015, TU Delft, Netherlands. Talk: *On low-rank updates of matrix functions.*
- Dobbiaco Summer School, *Matrix Theory and Computation with Applications to Network Analysis, Quantum Chemistry and Dynamical Systems*, June 15–20 2014, Dobbiaco, Italy.
- IWASEP10, June 2–5 2014, Dubrovnik, Croatia. Poster: *On low-rank updates of matrix functions.*
- Winter School on Hierarchical Matrices, 11–15 Mar 2013, Leipzig, Germany.