ON THE STRUCTURED EIGENVALUE PROBLEM: METHODS, ANALYSIS, AND APPLICATIONS

A Dissertation

Submitted to the Faculty

of

Purdue University

by

James P. Vogel

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

December 2018

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF DISSERTATION APPROVAL

To Melissa James, Katie Vogel, Pat Slaber, and Dick Vogel.

ACKNOWLEDGMENTS

I owe many thanks to the several individuals who helped me throughout graduate school. Firstly, I would like to thank my advisor and research supervisor, Professor Jianlin Xia. His consistent guidance over the past five years has been essential in my development as an applied mathematician. In my opinion, there is no better advisor of PhD students at Purdue than Professor Xia, and I will always be grateful to have been his student.

I am also grateful to my committee members for their continuous support: Prof. Ragu Balakrishnan, Prof. Jingwei Hu, Prof. Peijun Li, and Prof. Jie Shen. In particular, Prof. Ragu Balakrishnan, provided several semesters of financial support and gave a particularly interesting prospective on my research from his optimization and electrical engineering background. Some of the most important breakthroughs from my research came from talking to Professor Balakrishnan. In addition, Dr. Stephen Cauley, Prof. Ilse Ipsen, Prof. Xiaobai Sun, and Prof. Lieven Vandenberghe have also played an important role in my research.

I also would like to thank many fellow Purdue students who were great friends and colleagues over the past five years. Special thanks to Melissa James, Qinfeng Li, Difeng Cai, Duo Cao, Hee Jun Choi, David Imberti, Kyle Kloster, Lina Ma, Reggie McGee, Xiaofeng Ou, Kevin Rotz, Yingwei Wang, Yuanzhe Xi, Zixing Xin, Xin Ye, and Ziyao Yu.

Lastly, thank you to my wife, parents, and sister; who have supported whenever I needed them.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Vogel, James P. PhD, Purdue University, December 2018. On the Structured Eigenvalue Problem: Methods, Analysis, and Applications. Major Professor: Jianlin Xia.

This PhD thesis is an important development in the theories, methods, and applications of eigenvalue algorithms for structured matrices. Though eigenvalue problems have been well-studied, the class of matrices that admit very fast (near-linear time) algorithms was quite small until very recently. We developed and implemented a generalization of the famous symmetric tridiagonal divide-and-conquer algorithm to a much larger class of rank structured matrices (symmetric hierarchically semiseperable, or HSS) that appear frequently in applications. Altogether, this thesis makes valuable contributions to three different major areas of scientific computing: algorithmic development, numerical analysis, and applications. In addition to the previously stated divide-and-conquer algorithm, we generalize to larger classes of eigenvalue problems and provide several key new low-rank update algorithms. A major contribution the analysis of the structured eigenvalue problem. In addition to standard perturbation analysis, we elucidate some subtle and previously under-examined issues in structured matrix eigenvalue problems such as subspace contributions and secular equation conditioning. Finally, several applications are studied.

1. INTRODUCTION

As an illustration of the power of our techniques and how they fill a major need in the scientific computing community, consider the important problem of clinical magnetic resonance imaging (MRI). In one of the popular state-of-the-art methods for this problem, one combines multichannel compressed sensing with parallel imaging [16] and the result is a very large, dense (few zeros), and symmetric encoding matrix. In these methods (and in many image processing problems), the bottleneck step in terms of computational complexity and thus run time is performing a regularization optimization, which at its heart involves computing the set of all eigenvalues of the encoding matrix several times; where the diagonal of this matrix is updated at each step of the regularization.

There are several ways to solve for these eigenvalues. Given that matrix is large and dense, the algorithm to do this must be very general. For instance, consider the vast set of eigenvalue algorithms in Demmel's and Golub & Van Loan's influential numerical linear algebra textbooks [22, 33]. The only algorithms in this large collection that would work for the above regularization problem all require the use of either reduction of the matrix to tridiagonal form, trace-minimization, or an LDL^T factorization and subsequent inertia-based bisection scheme. Each of these sub-routines scales with $O(n^3)$ time complexity and $O(n^2)$ storage complexity, and thus are prohibitively slow for large matrices.

Moreover, they do little or nothing to utilize the plethora of structure in the matrices endemic to this image processing problem. As shown in [16], these encoding matrices have hierarchical rank structure, and can be well-approximated by hierarchical semiseperable (HSS) matrices. This class of matrices will be formally defined in following chapter, but heuristically the meaning both physically and algebraically is straightforward. Physically, this corresponds to a system of n nodes where the interac-

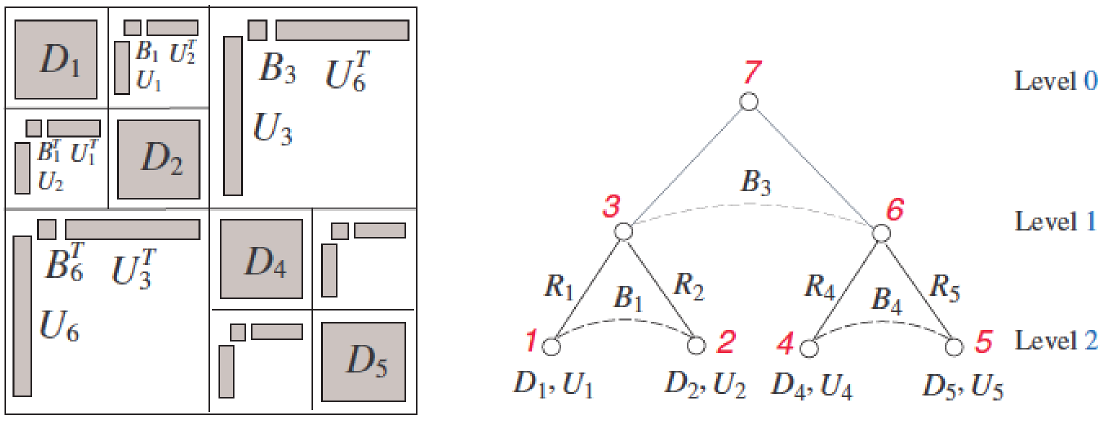


Fig. 1.1. The above figure is reprinted from [90]; it shows the HSS tree for a given symmetric HSS matrix

tions between far-away or "far-field" nodes are much weaker than so-called "near-field" interactions, and thus they can be accurately approximated with little computational expense, much in the spirt of fast-multipole methods (FMM) [6, 35].

Algebraically, they correspond to matrices where each off-diagonal block has a rank r (relative to a tolerance τ) which is small compared to the matrix size n. To be HSS, a matrix must also fulfill a hierarchical nesting property which allows it to admit a nice tree structure and very fast operations. We express and discuss this condition in the following chapter. Our algorithms take advantage of this structure to run in $O(r^2 n \log^2 n)$ time and with $O(rn \log n)$ storage.

While our algorithms are very fast, they are also extremely general and highly (provably) accurate. Returning to the MRI problem above, it would certainly be possible to customize a fast eigensolver for this problem (to our knowledge no such algorithm exists), but it would be of little use to those outside of the medical imaging community. The current state of "fast eigensolvers" finds itself in a relative stalemate where scientists and engineers lack the mathematical expertise to customize eigensolvers for their applications and the matrix algebra community lacks the man-hours

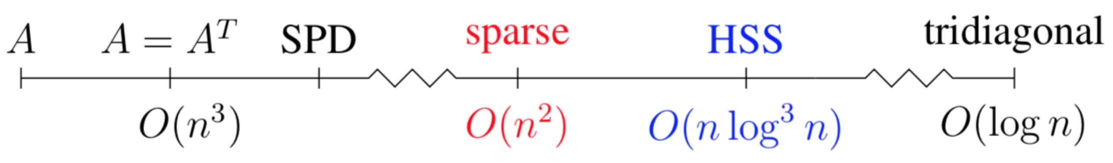


Fig. 1.2. A heuristic diagram of our general approximation strategy

to realistically derive algorithms for all of the important applications that rely upon eigenvalues and spectral decompositions.

The solution is to design algorithms in“the middle” (see Figure 1.2 below), for as large of a class of matrices as possible that still consistently and accurately admit algorithms in near-linear time and storage. We strongly believe that HSS matrices, and moreover the more general multi-layer hierarchically semiseparable matrix class (MHS) are the matrix classes that provide the most advantages in terms of the fast algorithms they admit, feasibility of rigorous analysis, suitability for high-performance implementations, and vast size that touches applications in nearly every field of science and engineering.

As one moves from right to left, the size of the class of matrices grows larger but the type of algorithms that are stable and accurate for that matrix class grow slower. We design algorithms for as large of classes of matrices as possible while still being near-linear time. There is a direct relationship between HSS matrices and tridiagonal or sparse matrices. This can be seen through the limit behavior of the eigenvectors as the off-diagonal rank r goes to 1 or to $n/2$, and as such it is a natural intermediate structure.

Returning again to our medical imaging example, it becomes clear that the accuracy of these eigenvalue algorithms can be an extremely important topic in applications. With thousands of sets of spectra computed during the regularization process [16], a seemingly small propagation of error could result in wild undulations in the reconstructed image. Given the context of a clinical MRI, to be accepted into practice and thus useful, our algorithms must demonstrate extremely rigorous stability and controllable accuracy results. Moreover, the structure of the decompositions and the nature of error must be completely understood in order to bound error in pathological cases. Even in far less life-and-death applications of eigensolvers, given the fundamental role played by eigendecomposition as subroutines of scientific processes, it is essential that they be rigorously analyzed in every aspect. This will be the biggest contribution of this thesis; gaining fundamental insight into the analytical nature of the structured eigenvalue problem.

Until recently, almost all near-linear time solvers (both for linear systems and for eigendecompositions) were either unstable numerically or unable to be proven stable. But within the past 10 years, work by Martinsson, Rokhlin, Xia, and others [49, 73, 91, 92, 95] have introduced stable near-linear time algorithms for structured linear systems and the pertinent analysis to prove it. In this thesis, we introduce several fast and stable eigensolvers for large classes of matrices and the supporting analysis. Previously, there were very few such eigensolvers. The most notable such algorithm is the symmetric tridiagonal divide-and-conquer algorithm of Gu and Eisenstat [39], which is provably stable and accuracy and runs in $O(n \log n)$ time and storage. It is currently very widely used and high-performance parallel implementations can be found in standard eigenvalue packages such as SCALAPACK and ANASAZI. This algorithm forms the basis of our work and will be introduced in the following chapter along with some classical eigenvalue analysis tools and results.

However, HSS (and moreover MHS) matrices are a much larger matrix class than tridiagonal. The smaller of the two classes, HSS, includes: banded matrices, Toeplitz matrices, the discretization of most elliptic PDEs, many problems from areas such as geophysics and electrical engineering, and can well-approximate matrices in countless more applications. The generalization from tridiagonal to HSS is highly nontrivial. In Chapter 3, we briefly discuss the HSSEIG algorithm which we introduced in our recent paper [88]. In Chapter 2 we give some of the recent developments in numerical linear algebra that allowed us to make such a novel generalization. These include hierarchical matrix techniques, randomized sampling, and structured perturbation analysis.

One notable feature about many of the algorithms in this thesis, buoying our ability to do rigorous analysis, is that many of the eigensolvers presented are so-called "direct" solvers, rather than iterative. The name is a slight-misnomer, as it can easily be shown using elementary Galois theory that any matrix of size larger than 4×4 will not in general have eigenvalues representable in floating point. This follows from the insolvability of the group A5, and in practice only the most trivial of matrices have eigenvalues that are exactly representable on computers. As such, it is a fact of life that all eigenvalue algorithms must have at least one iterative component.

The distinction then lies in where in the scheme of the algorithm the iterative component(s) occur. If the outer-most loop is an iterative process, the entire algorithm is referred to as iterative. This includes QR iterations, bisection, trace-min, and most eigensolver the reader is likely familiar with. However, in some special cases, most notably symmetric HSS and symmetric MHS matrices, we can perform an eigendecomposition where only one small and relatively simple subroutine is iterative. This can lead to many benefits, such as less reliance on convergence criteria and knowledge of the underlying structure. But the best benefit is that this greatly facilitates rigorous analysis of the algorithm. To better illustrate the idea of a "direct solver," we discuss the famous result of Golub [32] for the eigenvalues of a diagonal matrix plus a symmetric rank-one update. Let D be a diagonal matrix $\in \mathbb{R}^{n \times n}$ and $v \in \mathbb{R}^{n \times 1}$. We wish to

$$\text{find} \quad \{\lambda_i\} \quad \text{such that} \quad (D + vv^T)x = \lambda_i x \quad \text{for some} \quad x \in \mathbb{R}^{n \times 1}, \quad x \neq \overrightarrow{0} \tag{1.1}$$

To find the exact eigenvalues $\{\lambda_i\}$ satisfying (1.1), one would need to find the determinant $\det(D + vv^T - I\lambda)$. For a general matrix, analytically determining the determinant requires factorial time, which is infeasible computationally. This is why the eigenvalues are often approximated via an iterative method. But with a few lines of clever algebra, one can show that is determinant is equivalent to

$$f(\lambda) = 1 + \sum_{i=1}^{n} \frac{v_i^n}{d_i - \lambda}. \tag{1.2}$$

.

Without loss of generality (for we can deflate the problem if necessary), assume that $v_i \neq 0$ for $i = 1, 2, ..., n$, and that each eigenvalue of our original problem is unique, then by definition our eigenvalues are the values of λ satisfying

$$\det(D + vv^T - \lambda I) = 0.$$

We then note that

$$\begin{aligned}\det(D + vv^T - \lambda I) &= \det(D - \lambda I)\det(I + (D - \lambda I)^{-1} vv^T) \\ &= \prod_{i=1}^{n}(d_i - \lambda)\left(1 + \sum_{i=1}^{n} \frac{u_i^2}{(d_i - \lambda)}\right),\end{aligned}$$

and we have (1.2) above.

As will be shown in the following chapter, there are several methods to solve this equation very consistently, accurately, and efficiently. While an iterative method must ultimately be used for this subroutine, it can be a relatively simple, well-understood, and very well-analyzed iterative method. And if finding the zeros of (1.2) is the only iterative procedure in the entire eigenvalue algorithm (which is the case for many of the eigenvalue algorithms presented in this thesis), we are able to exercise a wonderful amount of control over the relevant analysis.

Returning once more to the the medical imaging application, we note that while HSSEIG can be applied to many regularization optimization encoding problems, in some cases the off-diagonal rank r is fairly large, around $\sqrt{n}$. This gives HSSEIG quadratic time-complexity instead of the aspired near-linear time (the complexity of this algorithm is described in more detail in Chapter 4). Thus, the clinical MRI example provided motivation to generalize our algorithms to even larger classes of structured matrices. One important such class is the aforementioned MHS matrices, recently introduced by Xia [94]. These have a multi-layer structure, which nests an HSS matrix within an HSS matrix, and in this way allows us to solve high-dimensional problems still in near-linear time. It is also possible to generalize HSSEIG to non-symmetric matrices and generalized eigenvalue problems, and Chapter 3 we discuss extending the HSSEIG algorithm non-symmetric HSS and generalized eigenvalue problems. This large variety of structured eigenvalue algorithms will allow problems from wide-ranging disciplines to be studied.

With this in mind, while we are excited that future work on MHS eigenvalue algorithms will allow us to solve important problems in medical imaging extremely quickly, accurately, and in a provably stable way, this is merely one example of the power of our work. The classes of matrices are extremely general, and thus their comprehensive analysis illuminates stability issues both within a plethora of their applications, and outside of them. By this, we mean that we can apply our eigensolvers as extremely fast preconditioners to general sparse problems, and given the surfeit of analysis on the eigenvalue perturbation and distributions, we will be able to give very rigorous bounds on how well the matrices have been preconditioned.

Finally, we note that these algorithms all have the potential for very high-performance implementations. Already, we have managed to implement HSSEIG in such a way that it scales like $O(n \log^3 n)$ for large classes of HSS matrices, and can be accurate to any user-specified tolerance, even machine precision in some cases. Indeed, there is likely much future work which can follow from this research program, as will be discussed in Chapter 6. This thesis seeks to lay a sound foundation; in each of algorithm development, numerical analysis, and applications, for that research program.

2. BACKGROUND MATERIAL

2.1 Foundations

At the very heart of structured eigenvalue computation is a very old idea: performing a rank-one update to a symmetric eigenvalue problem. We first review this in 2.1.1. In the section following we discuss how this idea was leveraged by Cuppen, Gu, and others in the 1980's and 1990's to design novel eigensolvers for symmetric tridiagonal matrices. As we seek more general algorithms, we briefly discuss the issues at play when generalizing to a multi-rank update in 2.1.3. Finally, in 2.1.4, we give some important classical eigenvalue perturbation theory and results, which we expand upon in our own analysis work in this thesis.

2.1.1 Fast rank-one update to the symmetric eigenproblem

An important underlying subroutine in many of the algorithms in this thesis is the rank-one update to the symmetric eigenproblem. This problem is well studied. We are able to consistently compute an update stably and accurately in $O(n)$ complexity. We follow the same algorithm as in [88] where details can be found, but here review the important concepts.

Suppose that A is a real and symmetric $n \times n$ matrix with a known eigendecomposition $A = Q\Lambda Q^T$. Further suppose that z is a real $n \times 1$ vectors and that we wish to solve for all eigenvalues of the symmetric matrix $\hat{A} = A + zz^T$. It is shown in [101] that this is equivalent to solving for the eigenvalues of the matrix

$$\Lambda + vv^T, \quad \text{where} \quad v = Q^T z. \tag{2.1}$$

This is a diagonal matrix plus rank-one update, and is very inexpensive computationally. It was shown in the previous section that solving for all eigenvalues of $\Lambda + vv^T$ is equivalent to solving for all zeros of the secular equation

$$f(\lambda) = 1 + \sum_{j=1}^{n} \frac{v_j^2}{\Lambda_{j,j} - \lambda}. \tag{2.2}$$

There are several ways to solve this equation efficiently. One of the most popular methods, which is employed often in the algorithms presented in this thesis, is to solve the secular equation via the rootfinder of Li [64]. The reason it is employed here is because it allows us to utilize the interlacing property of the eigenvalues and poles in the rank-one update problem to have superior stability properties. We are forced to employ specific stability checks for the final root, but as it is for one root this does not affect our overall complexity [88].

This root finder also admits acceleration via the Fast Multipole Method (FMM) [6, 35], which allows linear complexity time and storage. In the FMM, we have a function of the form

$$\varphi(\xi) = \sum_{j=1}^{n} c_j \varphi(\xi - \xi_j)$$

where $\varphi(x)$ is either $\log(x)$, $1/x$, or $1/x^2$, and we wish to evaluate the function at m points. While standard evaluation gives quadratic complexity, the FMM gives $O(m + n)$, while still being accurate to machine precision.

The eigenvectors of $\Lambda + vv^T$ also take on a simple form

$$x_j = \frac{(\lambda_j I_{n\times n} - \mathrm{diag}(\lambda))^{-1} v}{\|(\lambda_j I_{n\times n} - \mathrm{diag}(\lambda))^{-1} v\|}. \tag{2.3}$$

However, computing these vectors explicitly is an $O(n^2)$ operation, so the eigenvector matrix is seldom formed explicitly in efficient implementations of the divide-and-conquer algorithm. Fortunately the Cauchy-like form of these eigenvector matrices gives a convenient compressed storage.

One major issue that must be resolved to preserve the algorithm's stability is that if there exists λ, μ such that $|\lambda - \mu|$ is small, rapid loss of orthogonality will occur as shown in [24]. However, if as proposed in [39], we instead solve for the exact

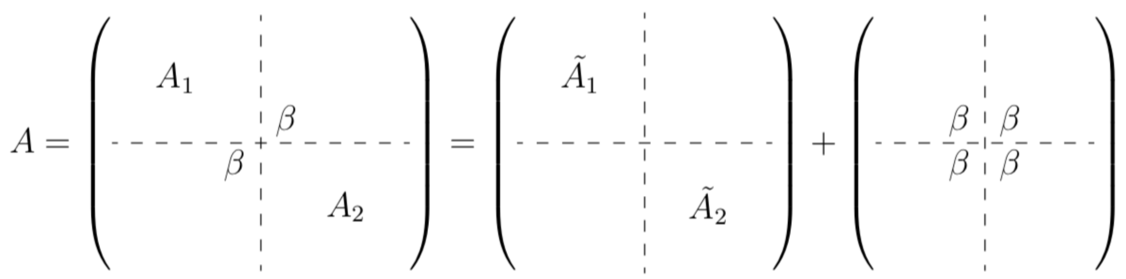


Fig. 2.1. A tridiagonal matrix split into blocks with a rank-one update

eigenvalues of a slightly perturbed problem $\Lambda + \hat{v}\hat{v}^T$, we have backwards stability and orthogonality is preserved. The vector $\hat{v}$ needs to computed with the following formula:

$$\hat{v}_k = \left(\frac{\prod_{j=1}^{k-1}(\Lambda_{k,k} - \lambda_j)\prod_{j=k}^{n}(\lambda_j - \Lambda_{k,k})}{\prod_{j=1}^{k-1}(\Lambda_{k,k} - \Lambda_{j,j})\prod_{j=k+1}^{n}(\Lambda_{j,j} - \Lambda_{k,k})} \right)^{-1/2} \tag{2.4}$$

In this way, we can consistently update the symmetric eigenproblem in a stable and efficient manner, and we utilize this frequently in our work.

2.1.2 The tridiagonal divide-and-conquer algorithm

These ideas can be used to efficiently solve for all eigenvalues of a large symmetric and real matrix the can be expressed as a hierarchy of diagonal (or more generally block diagonal) matrices and a set of rank-one (or more generally rank-r) updates. The simplest case of this is when the matrix is tridiagonal. Suppose that A is an $n \times n$ real, symmetric, and tridiagonal matrix.

$$A = \begin{pmatrix} a_1 & b_1 & & \\ b_1 & \ddots & \ddots & \\ & \ddots & an-1 & b_{n-1} \\ & & b_{n-1} & a_n \end{pmatrix}. \tag{2.5}$$

Then we can write A as the sum of a block diagonal matrix and a rank-one update, as in Figure 2.1.

Further suppose that the spectral decompositions $\tilde{A}_1 = Q_1\Lambda_1Q_1^T$ and $\tilde{A}_2 = Q_2\Lambda_2Q_2^T$ have been computed. Then we have that

$$A = \left(\begin{array}{c|c} Q_1 & \\ \hline & Q_2 \end{array}\right) \left[\left(\begin{array}{c|c} \Lambda_1 & \\ \hline & \Lambda_2 \end{array}\right) + \beta zz^T\right] \left(\begin{array}{c|c} Q_1^T & \\ \hline & Q_2^T \end{array}\right), \tag{2.6}$$

where $z^T = (q_1^T, q_2^T)$, with q_1^T the last row of Q_1 and q_2^T the first row of Q_2.

The matrices A_i, $i = 1, 2$ can also be divided recursively, finding the eigenvalues of $\tilde{A}_1$ and $\tilde{A}_2$ by further subdivisions with rank-one updates, and so on. In this was, an $n \times n$ eigenvalue problem can be reduced to a set of $m \times m$ eigenvalue problems with a set of rank-one corrections, where $m << n$ is a number small enough where the computational cost of solving the eigendecompositions of size $m \times m$ directly is small compared to the cost of the whole algorithm. This idea was first proposed in [21] and was formulated in a fast and stable way in [39]. The algorithm in [39] forms much of the theoretical basis for this thesis.

2.1.3 Fast multi-rank update to the symmetric eigenproblem

However, to fully generalize the work of Gu and and others, it is not sufficient to consider just rank-one updates. Instead, we must consider multi-rank (rank k) updates where $1 < r << n$, though in practice r may be on the order of several thousand for sufficiently large n. There has been significant work done in this area, but no existing algorithm had suitable accuracy, efficiency, and stability properties, so we opted to write our own. This multi-rank update algorithm is a significant portion of this thesis. In this section we briefly describe what make this subroutine difficult to perform in practice, what has been tried to solve it historically, and the weaknesses of the historical algorithms that we will improve upon.

One of them most common methods to solve a multi-rank update historically has been to break the rank-r update into r rank-one updates. Mathematically, we have that

$$\hat{A} = \Lambda + ZZ^T = \Lambda + \sum_{i=1}^{r} z_i z_i^T.$$

It is a known result [13, 22] that finding the eigenvalues of a rank-one update to a diagonal matrix is mathematical equivalent to solving for the roots of the secular equation

$$f(\lambda) = 1 + \sum_{j=1}^{n} \frac{v_i(j)^2}{\Lambda(j,j) - \lambda}. \tag{2.7}$$

This method of multi-rank update by multiple rank-one updates has $O(rn)$ computational complexity and is very accurate for well-separated problems, that is problems where the gaps between all eigenvalues is sufficiently large. However, for clustered eigenvalues, it has poor stability. This stability issue will be further discussed in subsequent chapters. Moreover, it is an inherently sequential process, while it may admit a pipelining process, a true distributed memory implementation would require a different approach. This method is highly recommended if a user has a problem they wish to solve sequentially on well-separated problems, but in other cases our new method may be a superior approach.

Before describing the second common family of methods, we point out how the intuitive generalization approach does not work in this case. By this, we mean we wish to examine how the secular equation is derived, and if a similar approach could be used to derive a more general secular equation for a multi-rank update. It is shown in several places such as [22] that the secular equation is derived from a simple determinant expansion. If one takes this process one step further, another satisfactory result is achieved. It is shown in a few places such as [2] that the secular equation for a rank-two update is

$$f(\lambda) = \sum_{q=1}^{n} \sum_{r=q+1}^{n} \frac{(v_{1q}v_{2r} - v_{1r}v_{2q})^2}{(\lambda - d_q)(\lambda - d_r)} - \sum_{q=1}^{n} \frac{v_{1q}^2 + v_{2q}^2}{\lambda - d_q} + 1. \tag{2.8}$$

However, if we examine the above equation and consider what the determinant expansion of a general multi-rank update would look like, we see that the number of terms is $O(r!n)$, not $O(rn)$ as one might hope. With this in mind, the approach is inferior to most standard eigendecomposition techniques, and cannot be used in the multi-rank case.

The second popular approach are called inertia-bisection based approaches. These rely on Sylvester's Inertia Theorem [72], which tells us that for any symmetric matrix A and invertible matrix L, the inertia or number of positive and negative eigenvalues is the same for A and for D with $A = LDL^T$. Often for a low-rank update an LDL^T factorization is much cheaper to compute than an eigendecomposition. This can be exploited to quickly evaluate the inertia of A with many shifts, and thus locate the eigenvalues using a bisection scheme. In the presence of further structure, it is shown such as in [90] that an inertia-bisection approach can be done in a fast and structured way. But there is a natural bottleneck in this approach of $O(n^2)$ computational complexity, and therefore it is unsatisfactory for our purposes.

In this thesis, we seek an algorithm that both is computationally superfast, or in nearly linear time complexity, but also has excellent data locality for parallelization and remains stable in the presence of clusters. To do this we use a fundamentally different approach in using a quasi-Newton optimization approach, with highly structured formulations and algebraic expansions to increased efficiency and an inverse eigenvalue problem to preserve orthogonality. We will say much more about quasi-Newton methods and how to effectively compute a multi-rank update in Chapter 3. Moreover, Chapter 4 will give detailed convexity and complexity analyses of our new methods.

2.1.4 Classical eigenvalue perturbation analysis

The comprehensive analysis proposed for this thesis will rely heavily on novel new analysis techniques for the matrix eigenvalue problem, but these techniques all stem from and take their spirit from the rich literature of classical eigenvalue perturbation analysis. And in fact, a result from almost 70 years ago is used notably in our preliminary analysis. This is Weyl's Theorem [102], which states that if M, H, and P are symmetric matrices with $M = H + P$, where the eigenvalues of these three matrices are respectively $\{\lambda_1 \geq ... \geq \lambda_n\}, \{\nu_1 \geq ... \geq \nu_n\}$, and $\{\rho_1 \geq ... \geq \rho_n\}$, then:

$$\nu_i + \rho_n \leq \lambda_i \leq \nu_i + \rho_i \quad \text{for all} \quad i = 1, ..., n$$

It is useful to use as it gives a rough but universally applicable bound on the shift of eigenvalues after a symmetric rank-one update, a subroutine frequently performed in our algorithms. Another extremely powerful eigenvalue analysis theorem that we often employ is due to Sylvester [80]. This theorem states that for any invertible matrix P such that $PAP^T = S$, then the number of eigenvalues with positive, zero, or negative parity respectively is the same for matrices P and S. This so-called "Inertial Theorem" has powerful ramifications for fast bisection methods that still prove very useful to us. There is an excellent collection of classical results such as these in Wilkinson's monograph on the matrix eigenvalue problem [101].

More still, there are several important results from the later 20th century that look slightly deeper in the structure of matrices to develop tighter bounds on eigenvalue perturbation. Many such as Paige [69] showed how in the case of well-separated eigenvalues, the effects of perturbations will be lessened and problems are more stable. Several papers such as [7, 10] utilized the notion of invariant subspaces to better analyze the spectrum. This technique can be generalized and will be used extensively in this thesis. The textbooks of Parlett [72] and Watkins [100] give a complete recounting of the important classical perturbation results for the symmetric and non-symmetric eigenvalue problems respectively.

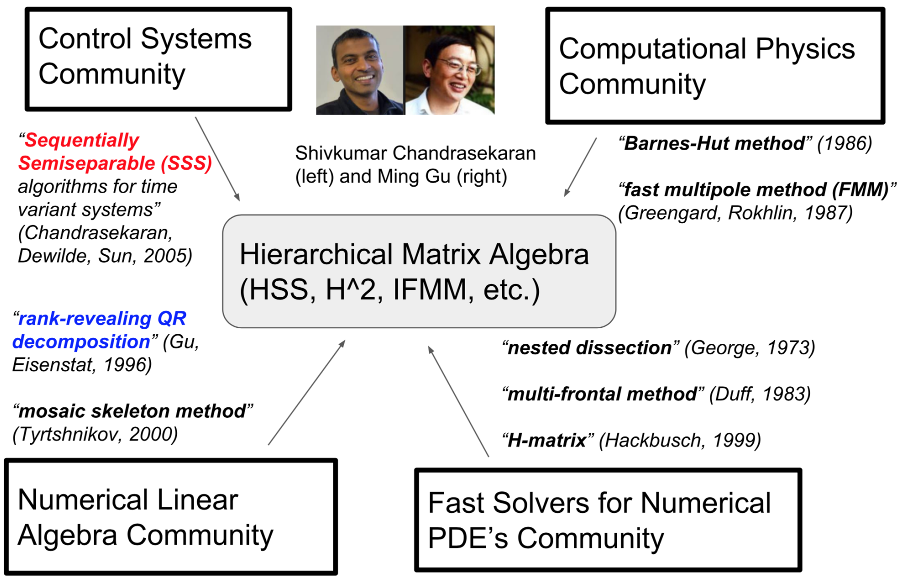

Control Systems Community
Computational Physics Community
Shivkumar Chandrasekaran (left) and Ming Gu (right)
"Sequentially Semiseparable (SSS) algorithms for time variant systems" (Chandrasekaran, Dewilde, Sun, 2005)
"Barnes-Hut method" (1986)
"fast multipole method (FMM)" (Greengard, Rokhlin, 1987)
Hierarchical Matrix Algebra (HSS, H^2, IFMM, etc.)
"rank-revealing QR decomposition" (Gu, Eisenstat, 1996)
"mosaic skeleton method" (Tyrtshnikov, 2000)
"nested dissection" (George, 1973)
"multi-frontal method" (Duff, 1983)
"H-matrix" (Hackbusch, 1999)
Numerical Linear Algebra Community
Fast Solvers for Numerical PDE's Community

Fig. 2.2. Predecessors and precursors to the HSS matrix

2.2 Recent developments

In this section we give some more recent theoretical and computational developments in the field of numerical linear algebra that have had a direct impact on this thesis work. The two most important are those of hierarchically semiseparable (HSS) matrices and randomized techniques. We also briefly mention some recent developments in parallel computing here. Parallel numerical linear algebra has been around for decades, but there are some special nuances that arise when applying parallelism to randomized or hierarchical algorithms.

2.2.1 Hierarchically semiseparable matrices (HSS)

The notion of a hierarchical matrix dates back to the 1980's. The earliest variants were those developed by particle physicists for the n-body problem, who noticed that far-field interactions were weaker and needed less information to be represented accurately. Notable early work on this was done by Barnes and Hut [5, 25], with a major breakthrough being the Fast Multipole Method (FMM) of Greengard and Rokhlin [6, 35], which in many ways laid the foundation for hierarchical matrix computations for years to come.

Many aspects of hierarchical matrix algebra, such as nested dissection [30] and the multi-frontal technique [26], come from the computational partial differential equation (comp. PDE) community. Wolfgang Hackbusch, considered by many to be the founder of the modern hierarchical matrix algebra research area, had previously done pioneering work in comp. PDE's through his work in multi-grid methods [42]. Around the turn on the 21st century, Hackbusch, introduced the numerical linear algebra community to the $\mathcal{H}$-matrix [43], which combined hierarchical matrices with the formalism of an algebra and was one of the biggest developments in the history of hierarchical matrices. $\mathcal{H}-$ matrices were studied extensively in the first ten years of the 21st century [44, 46].

Hierarchical matrix algebra also relies heavily on methods of algebraic compression, and there was much work in this area by the numerical linear algebra community in the 1990's. Some pioneering work in this area was done by Tyrtshnikov [83, 84]. Subsequently Gu and Eisenstat introduced the rank-revealing QR decomposition [40], which became ubiquitous in hierarchical matrix computations. It is still widely in use albeit in new randomized forms.

A final precursor to the HSS formalism we study is the separable nesting property of matrix row and column basis. This property occurs naturally in matrices arising in many applications, but perhaps nowhere is it more pronounced than in control problems arising from electrical engineering applications. These problems were efficiently solved by sequentially semiseparable matrix representations by Chandraskekaran and others [18].

Finally, this leads us to the hierarchically semiseparable (HSS) matrix representation, which is the one mostly studied in this thesis. It is formally defined below. We should also note there are other popular nested hierarchical matrix alternatives to the

HSS form currently in use. The two most notable being the $\mathcal{H}^2$ [45,47] matrix and the inverse fast multipole method (IFMM) [1]. We choose to focus on the HSS formalism here due to its superior stability properties, amongst other convenient features.

Definition 2.2.1 Hierarchically Semiseparable Matrix: *Suppose that $\mathcal{T}$ is a full postordered binary tree with $2k-1$ nodes. Further suppose that A is an $n \times n$ real matrix defined hierarchically such that at each node i of $\mathcal{T}$ there exist matrices D_i, U_i, V_i, R_i, W_i, and B_i (called HSS generators) satisfying the following relations for $i = 1, 2, ..., 2k-1$:*

$$D_i \cong A|_{t_i \times t_i} = \begin{pmatrix} D_{c_1} & U_{c_1} B_{c_1} V_{c_2}^T \\ U_{c_2} B_{c_2} V_{c_1}^T & D_{c_2} \end{pmatrix},$$

$$V_i = \begin{pmatrix} V_{c_1} & \\ & V_{c_2} \end{pmatrix} \begin{pmatrix} W_{c_1} \\ W_{c_2} \end{pmatrix}, \quad U_i = \begin{pmatrix} V_{c_1} & \\ & V_{c_2} \end{pmatrix} \begin{pmatrix} R_{c_1} \\ R_{c_2} \end{pmatrix}.$$

Let

$$A_i^- = \begin{pmatrix} A|_{t_i \times t_i^c} & A|_{t_i \times t_i^r} \end{pmatrix}, \quad A_i^| = \begin{pmatrix} A|_{t_i^c \times t_i} \\ A|_{t_i^r \times t_i} \end{pmatrix}.$$

The blocks A_i^- and $A_i^|$ are called the HSS block row and column associated with node i, respectively.The maximum numerical rank of al HSS blocks is called the HSS rank of A, denoted by r. Note that for HSS matrices, the optimal size for the leaf-level blocks is approximately $2r$, where r is the HSS rank of the matrix. In the case that A is a symmetric matrix, we also have the relations that $U_i = V_i$ for all i, $R_i = V_i$ for all i, and $B_i = B_j^T$, where node j is the sibling of node i.

Algorithms have recently been developed by Gu, Xia, and others that compute HSS matrix factorization, multiplication, inversion, etc. in a way that is both stable, very accurate, and extremely fast [91–93, 95]. However, until very recently, there was little literature on how to quickly compute the eigendecomposition of an HSS matrix. Recently, a novel algorithm based on a fast LDL factorization and an inertia-based bisection scheme was introduced [90] and used to find a small subset of the eigenvalues of an HSS matrix in O(n) time or all eigenvalues in $O(n^2)$ time. While this was an

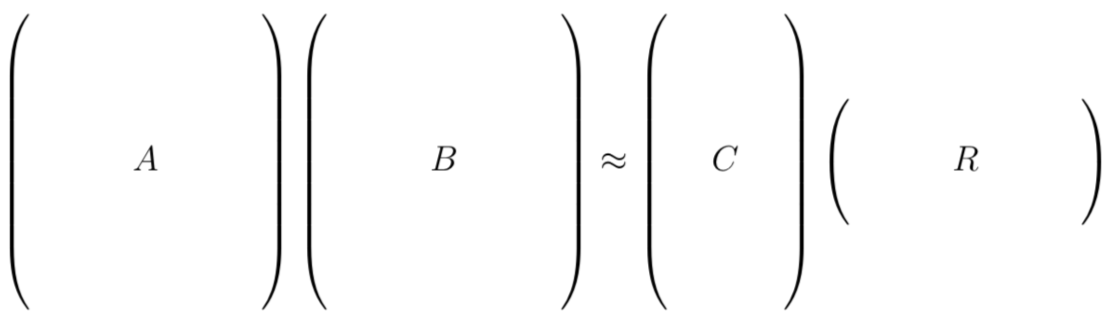


Fig. 2.3. Heuristic illustration of a basic randomized sampling procedure

extremely important step in the evolution of structure eigenvalue algorithms, it fails to achieve near-linear complexity, and also does not give gives the eigenvectors of the HSS matrix or various other spectral decompositions. Still, the analysis in [90] provides many interesting ideas which we expand upon in this thesis.

2.2.2 Randomized sampling

Randomized sampling is an extremely valuable tool-box within numerical linear algebra, that has been around for a long time but is only now starting to be well-understood and be rigorously analyzed. The main idea is that if we have massively large matrices such that they are either too large to store in memory or very slow to store, then if we carefully sample a subset of the rows or columns (or more generally blocks or elements), then we can construct a new smaller matrix that is close to the original matrix with respect to norms with high probability.

This idea was first introduced over 60 yeas ago by Ulman and Von Neumann [27]. It is now used widely in many areas of numerical linear algebra, such as machine learning [60], high-performance computing [3], and theoretical computer science [23]. Recently, it has been extended to aid in the fast (near-linear time) construction of hierarchical forms (such as HSS) to well-approximate a matrix. This was first introduced by Rokhlin [69], and there is a very nice survey by Halko, Martinsson, and Tropp [43] which illustrates some of the novel ways randomized sampling can be applied to hierarchical matrix algebra problems.

In the case of HSS matrices, it was shown in [97] that a Toeplitz matrix admits an HSS construction in near-linear time. This was further extended to work on fast eigensolvers in [90]. The eigenvalue algorithms in [90] are $O(n^2)$ time complexity, but they do well to illustrate the power of randomization in rank-structured eigenvalue algorithms. Most notably, a recent paper by Gu [38] utilizes tools from randomized sampling and invariant subspaces to quickly compute a Singular Value Decomposition (SVD) of a matrix, if only the top part of the spectrum is required.

Perhaps the most notable recent development in randomized sampling is in its analysis, and we will make lots of use of these developments in this thesis. This analysis centers largely around so called "concentration inequalities" which provide probabilistic bounds on how a random variable deviates from its expectation. Common concentration inequalities include Markov's inequality, Chebyshev's inequality, and Bernstein's inequality. The latter plays in a very important role in randomized numerical linear algebra, and thus we state a special case useful to our work. Let $X_1, ..., X_m$ be independent Bernouli random variables taking values ± 1 with equal probability. Then for every $\epsilon > 0$:

$$P\{|\frac{1}{n}\sum_{i=1}^{n} X_i| > \epsilon\} \leq 2\exp\{-\frac{n\epsilon^2}{2(1+\epsilon/3)}\}. \tag{2.9}$$

Using these types of concentration inequalities and their generalizations, many authors have recently been able to prove powerful results on the high success probabilities of their stochastic algorithms, essentially allowing users to treat their codes as deterministic; albeit it much faster [11,20,52]. We will make use of these ideas and expand upon them throughout this thesis.

2.2.3 Modern parallel computing

Though parallel computing has been around for decades, it is still a rapidly developing field. An excellent recently published text by Gallopoulos, Phillipe, and Sameh [29] does very well to illustrate the current state of parallel computing, to highlight the recent development, and to elucidate the major strategies by which these types of algorithms can be analyzed. We make extensive use of this reference. The aforementioned tridiagonal divide-and-conquer algorithm, which forms the basis of our work, has been effectively parallelized for over a decade [81].

Moreover, the HSS structures targeted by many of our algorithms also have huge potential to be parallelized. In the cases of parallel HSS construction, parallel ULV factorization, and parallel HSS linear system solutions, this has already been done [99]. Moreover, such algorithms are currently being developed for distributed memory environments [74, 98]. This gives us much reason to be believe that our eigensolvers will enjoy similar parallizability properties. These papers also perform significant analysis of the communication costs of these parallel algorithms, giving us a natural benchmark when performing parallel analysis on our high-performance eigensolvers.

We pay particular attention to the study of process grids for HSS matrices in [98]. They consider the relationship between different process grids due to the interaction between parent and children grids. In the case that $G_i = G_{c_1} \cup G_{c_2}, |G_{c_1}| = |G_{c_2}|$, the children process grids can have the same shape and the parent grid can be generated by combining the two smaller grids. This previous work in parallel HSS algorithms has reduced the redistribution cost to a pairwise exchange. This choice of process groups demonstrates a pursuit of utilizing every processing unit. An alternative approach reduces the redistribution cost into a pairwise exchange type as well. This choice bounds the the communication cost in each group while allowing some idling. In our implementation, we will combine the two techniques, and this will give more control over the size of the process grid on each level.

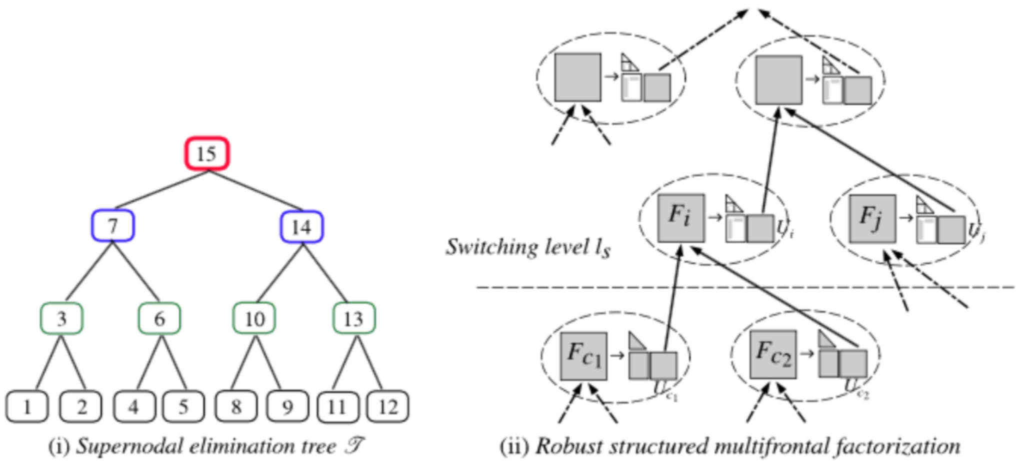

(i) Supernodal elimination tree $\mathcal{T}$
(ii) Robust structured multifrontal factorization
Switching level l_s
F_i
F_j
F_{c_1}
F_{c_2}
U_i
U_j
U_{c_1}
U_{c_2}

Fig. 2.4. The above figure is reprinted from from [95]; it shows an illustration of a switching level (here denoted by l_s) in the context of supernodal elimination

Much or our high-performance computing work focused on implementing and analyzing adaptive processes for detecting and utilizing matrix structure with however much or little information about the matrix the user provides. A lot of work in this area has already been done, mostly in the context of randomized HSS construction such as in [90]. These papers show that we do not *a priori* require any information about the rank-structure of a matrix in order to construct a compact HSS or MHS representation.

A particularly important idea is that of the switching level. This concept is that within an HSS tree, at lower levels it will be faster to use standard numerical linear algebra techniques but at higher levels it will be much faster to use hierarchical techniques. Determining where the optimal location to put this switching level is a major factor in optimizing any HSS or MHS algorithm. We illustrate this idea with the above picture, an example from [95].

Switching levels play a vital role in the algorithms introduced throughout this thesis. While no parallel implementation is included in this work, one is in progress and results will be communicated through future work. In the future, in addition to rudimentary parallelization, it may also be possible to do more sophisticated modern computing techniques such as through the use of GPUs, communication-avoiding algorithms, error-resistant algorithms, and more adaptive implementations. Moreover, as the field of high-performance computing continues to develop this research program will make sure to utilize the latest cutting edge tools and trends.

3. METHODS FOR THE STRUCTURED EIGENVALUE PROBLEM

3.1 Superfast divide-and-conquer eigenvalue algorithm

The basis for our divide-and-conquer algorithm is the tridiagonal divide-and-conquer algorithm discussed in the introduction. As we describe the generalization in detail in this section, we first succinctly state the full algorithm below so that it can be referenced compared to the superfast divide-and-conquer below. Note that in the divide stage, instead of a trivial subtraction of individual matrix elements at each tree node, we must transform a set of B and D generators as each node is visited. There is a similar increase in problem difficulty in the conquer. Instead of performing a sequence of single-rank updates, we must perform a sequence of multi-rank updates. But first we discuss a vitally important implicit pre-processing step. It is obvious that a tridiagonal matrix is already in a native data structure that allows for fast computation. However, in contrast, the full HSS case must often be transformed to a nearby problem to take advantage of data sparsity.

3.1.1 Construct

The algorithm can be used to quickly compute the eigendecomposition of matrices with the low-rank property. These types of matrices arise in various fields, and their HSS forms (or approximations of these forms) can be constructed with a variety of different strategies. If the numerical values of the individual matrix entries is the extent of what we know about the matrix, then a direct HSS construction [95] may be used. In practice, this is usually unnecessary. Often, fast algebraic or analytical methods can be employed for the construction of the HSS matrix, and the cost is about

Algorithm 1 Tridiagonal divide-and-conquer

```
1: procedure (Input: matrix $A$; output: eigenvalue vector $\lambda$)
2:     for level $i$ from 0 to $s-1$ do                                  ▷ $s \leq \log_2 n$
3:         for block diagonal matrices $j$ from 1 to $2^i$ do
4:             Partition $A_j = \begin{pmatrix} A_{j,1} & 0 \\ 0 & A_{j,2} \end{pmatrix} + \beta z z^T$
5:             Store scalar $\beta$
6:         end for
7:     end for
8:     for block diagonal matrices $j$ from 1 to $2^s$ do
9:         Directly solve for eigendecomposition of $j$ such that $A_j = Q_j \Lambda_j Q_j^T$
10:    end for
11:    for level $i$ from $s-1$ to 0 do
12:        for block diagonal matrices $j$ from 1 to $2^i$ do
13:            Form $\Lambda + \beta z z^T$ from $Q$'s, $\Lambda$'s, $\beta$ from previous level
14:            Solve secular equation to obtain eigenvalues $\Lambda$ of $(\Lambda + \beta z z^T)$
15:            for eigenvalue $k$ from 1 to size($Q$) do
16:                Column $k$ of $Q' = \frac{(\lambda_k I - \Lambda)^{-1} z}{\|(\lambda_k I - \Lambda)^{-1} z\|}$
17:            end for
18:            Form last or first row of $Q = \begin{pmatrix} Q_1 & 0 \\ 0 & Q_2 \end{pmatrix} Q'$   ▷ if $j$ is odd/even resp.
19:        end for
20:    end for
21:    $\lambda = \text{diag}(\Lambda)$
22: end procedure
```

This algorithm is $O(n^2)$ as shown or $O(n \log n)$ if accelerated with the FMM. The most expensive steps are 14, 16, and 18, which are all $O(n^2)$ or $O(n \log n)$ with FMM.

$O(n)$ or less. For example, for banded matrices, an HSS form can be constructed adaptively. For Toeplitz matrices, the HSS construction can be done in nearly $O(n)$ flops with the help of randomized methods. These are explained as follows.

If A is banded with blocks A_{jj} on the main diagonal and $A_{j,j+1}$ on the first block superdiagonal, then the HSS generators look like [91]

$$D_{\mathbf{i}} = \begin{pmatrix} A_{j-1,j-1} & A_{j-1,j} & \\ A_{j,j-1} & A_{j,j} & A_{j,j+1} \\ & A_{j+1,j} & A_{j+1,j+1} \end{pmatrix}, \quad U_i = \begin{pmatrix} I & 0 \\ 0 & 0 \\ 0 & I \end{pmatrix},$$

$$R_{\mathbf{c}_1} = \begin{pmatrix} I & 0 \\ 0 & 0 \end{pmatrix}, \; R_{\mathbf{c}_2} = \begin{pmatrix} 0 & 0 \\ 0 & I \end{pmatrix}, \quad B_{\mathbf{c}_1} = \begin{pmatrix} 0 & 0 \\ A_{j+1,j+2} & 0 \end{pmatrix},$$

where the zero and identity blocks have sizes bounded by the half bandwidth. Thus, the bandwidth of A determines its off-diagonal rank bound r.

Our algorithm can also be applied to Toeplitz matrices, and may be modified for other structured matrices (Toeplitz-like, Hankel, and Hankel-like) with the aid of displacement structures [31, 37, 50, 63, 70]. In fact, the rank structure of Toeplitz matrices in Fourier space is known as follows.

Theorem 3.1.1 *[61, 71] For a Toeplitz matrix A, let C be a Cauchy-like matrix resulting from the transformation of T into Fourier space through the use of displacement structures. Then the off-diagonal numerical ranks of C are $O(\log n)$ for a given tolerance.*

In particular, to preserve the symmetry as well as the real entries [61], we use the following Cauchy-like form:

$$C = \mathcal{F}_n A \mathcal{F}_n^*, \tag{3.1}$$

where $\mathcal{F}_n$ is the order-n normalized inverse discrete Fourier transform matrix. C can be approximated by an HSS form via a randomized HSS construction [90]. This construction is based on fast Toeplitz matrix-vector multiplication and randomized low-rank approximation, and costs $O(n \log^2 n)$.

For applications involving simple discretized kernel matrices, multipole expansions may be used to construct the HSS form [14].

For practical problems such as Toeplitz matrices, a dense matrix A is approximated by an HSS form $\tilde{A}$ first. We thus study the impact of off-diagonal compression on the accuracy of the eigenvalues and verify that the accuracy is well controlled by the approximation tolerance (and the FMM accuracy which, as an implementation issue, can be made very high and is not discussed). The study can be viewed as structured perturbation analysis for Hermitian eigenvalue problems. Previously, for special cases such as tridiagonal or banded A, there have been various studies on whether a small off-diagonal entry or block can be neglected [48, 54, 57, 69, 103]. Here for dense A, we are only truncating the singular values of the off-diagonal blocks. A significant benefit of an HSS approximation is to enable us to conveniently assess how the off-diagonal compression affects the accuracy of the eigenvalues.

3.1.2 Divide

An $n \times n$ symmetric HSS matrix A has the following form [18, 95]. Let $\mathcal{T}$ be a full postordered binary tree with $\mathbf{k}$ nodes $\mathbf{i} = 1, 2, \ldots, \mathbf{k}$, and each node $\mathbf{i}$ is associated with a contiguous index set $t_{\mathbf{i}}$ that satisfies $t_{\mathbf{k}} \equiv \{1 : n\}$ and $t_{\mathbf{i}} = t_{\mathbf{c}_1} \cup t_{\mathbf{c}_2}$, $t_{\mathbf{c}_1} \cap t_{\mathbf{c}_2} = \emptyset$ for each nonleaf node $\mathbf{i}$ with children $\mathbf{c}_1$ and $\mathbf{c}_2$ and $\mathbf{c}_1 < \mathbf{c}_2$. The matrix A is in a symmetric HSS form if there exist matrices $D_{\mathbf{i}}$, $U_{\mathbf{i}}$, $R_{\mathbf{i}}$, and $B_{\mathbf{i}}$ (called HSS generators) corresponding to each node $\mathbf{i}$ of $\mathcal{T}$, such that

$$A|_{t_{\mathbf{i}} \times t_{\mathbf{i}}} \equiv D_{\mathbf{i}} = \begin{pmatrix} D_{\mathbf{c}_1} & U_{\mathbf{c}_1} B_{\mathbf{c}_1} U_{\mathbf{c}_2}^T \\ U_{\mathbf{c}_2} B_{\mathbf{c}_1}^T U_{\mathbf{c}_1}^T & D_{\mathbf{c}_2} \end{pmatrix}, \tag{3.2}$$

$$U_{\mathbf{i}} = \begin{pmatrix} U_{\mathbf{c}_1} & \\ & U_{\mathbf{c}_2} \end{pmatrix} \begin{pmatrix} R_{\mathbf{c}_1} \\ R_{\mathbf{c}_2} \end{pmatrix}, \tag{3.3}$$

where $A|_{t_{\mathbf{i}} \times t_{\mathbf{j}}}$ denotes a submatrix of A selected by the row index set $t_{\mathbf{i}}$ and column index set $t_{\mathbf{j}}$. Clearly, $U_{\mathbf{i}}$ is a basis matrix of an off-diagonal block (where we assume U_i has full column rank). It is usually said to be a nested basis (matrix). Since $B_{\mathbf{c}_2} = B_{\mathbf{c}_1}^T$, we usually do not mention $B_{\mathbf{c}_2}$.

For notational convenience, we make the following assumptions:

- unless otherwise specified, we use $\mathbf{c}_1$ and $\mathbf{c}_2$ for the left and right children of a nonleaf node $\mathbf{i}$, respectively;
- assume the rank of each off-diagonal block $U_{\mathbf{c}_1} B_{\mathbf{c}_1} U_{\mathbf{c}_2}^T$ is (bounded by) r; more specifically, the order of $B_{\mathbf{c}_1}$ is (bounded by) r;
- $\mathrm{par}(i)$ and $\mathrm{sib}(\mathbf{i})$ denote the parent and the sibling of a node $\mathbf{i}$ in the HSS tree $\mathcal{T}$, respectively;
- $\mathbf{k}$ is the root of $\mathcal{T}$.

Let

$$A_i^- = \begin{pmatrix} A|_{t_i \times t_i^c} & A|_{t_i \times t_i^r} \end{pmatrix}, A_i^| = \begin{pmatrix} A|_{t_i^c \times t_i} \\ A|_{t_i^r \times t_i} \end{pmatrix}. \tag{3.4}$$

The blocks A_i^- and $A_i^|$ are called the HSS block row and column associated with node i, respectively. The maximum numerical rank of all HSS blocks is called the HSS rank of A, denoted by r in this paper. Note that for HSS matrices, the optimal size for the leaf-level blocks is $2r$ [92], where r is the HSS rank of the matrix. As such, this is the block size we used in our algorithm.

In our superfast DC method, the HSS matrix is divided into a block diagonal matrix (with smaller HSS diagonal blocks) plus a low-rank update. The eigenvalues and eigenvectors are recursively computed, with the major computations accelerated by FMM.

In the "dividing" stage, we recursively write the HSS matrix A as the sum of a block diagonal matrix (with two HSS diagonal blocks) plus a rank-r update.

Let $\mathbf{i}$ be a nonleaf node of $\mathcal{T}$, and DC is applied to $D_{\mathbf{i}}$. If $\mathbf{i} = \mathbf{k}$, then this is to divide the overall matrix A. It is clear that we can rewrite in the following form:

$$D_{\mathbf{i}} = \begin{pmatrix} D_{\mathbf{c}_1} & \\ & D_{\mathbf{c}_2} \end{pmatrix} + \begin{pmatrix} U_{\mathbf{c}_1} & \\ & U_{\mathbf{c}_2} \end{pmatrix} \begin{pmatrix} & B_{\mathbf{c}_1} \\ B_{\mathbf{c}_1}^T & \end{pmatrix} \begin{pmatrix} U_{\mathbf{c}_1}^T & \\ & U_{\mathbf{c}_2}^T \end{pmatrix}.$$

If we further compute an eigendecomposition of $\begin{pmatrix} & B_{\mathbf{c}_1} \\ B_{\mathbf{c}_1}^T & \end{pmatrix}$, this would result in a rank-$2r$ update to $\mathrm{diag}(D_{\mathbf{c}_1}, D_{\mathbf{c}_2})$. However, it turns out that we can write a more compact low-rank update, instead:

$$D_{\mathbf{i}} = \mathrm{diag}(\tilde{D}_{\mathbf{c}_1}, \tilde{D}_{\mathbf{c}_2}) + Z_{\mathbf{i}} Z_{\mathbf{i}}^T, \tag{3.5}$$

where

$$\tilde{D}_{\mathbf{c}_1} = D_{\mathbf{c}_1} - U_{\mathbf{c}_1} U_{\mathbf{c}_1}^T, \quad \tilde{D}_{\mathbf{c}_2} = D_{\mathbf{c}_2} - U_{\mathbf{c}_1} B_{\mathbf{c}_1}^T B_{\mathbf{c}_1} U_{\mathbf{c}_2}^T, \tag{3.6}$$
$$Z_{\mathbf{i}} = \begin{pmatrix} U_{\mathbf{c}_1} \\ U_{\mathbf{c}_2} B_{\mathbf{c}_1}^T \end{pmatrix}.$$

That is, by modifying the diagonal blocks, we can write $D_{\mathbf{i}}$ as a rank-r update to $\mathrm{diag}(\tilde{D}_{\mathbf{c}_1}, \tilde{D}_{\mathbf{c}_2})$. Note that this is much more preferable, since the diagonal blocks can be quickly updated with our scheme below. On the other hand, the later conquering stage is usually a much more expensive process, and a rank-$2r$ update would double its cost.

A critical issue is then to preserve the HSS structure in the dividing step, so that the HSS ranks of $\tilde{D}_{c_1}$ and $\tilde{D}_{c_2}$ do not increase. In fact, the HSS forms of $\tilde{D}_{\mathbf{c}_1}$ and $\tilde{D}_{\mathbf{c}_2}$ can be quickly updated based on those of $D_{\mathbf{c}_1}$ and $D_{\mathbf{c}_2}$, respectively. This follows from the property of the nested basis $U_{\mathbf{i}}$. Similar structure updates have been exploited previously in HSS factorization and inversion [93, 95, 96]. Here, we show how to perform the HSS update in a more intuitive way.

Theorem 3.1.2 *For the nested basis $U_{\mathbf{i}}$ associated with node $\mathbf{i}$, let H be a square matrix with size equal to the column size of $U_{\mathbf{i}}$. Then $U_{\mathbf{i}} H U_{\mathbf{i}}^T$ is an HSS matrix, with the HSS generators $\hat{D}_{\mathbf{j}}, \hat{U}_{\mathbf{j}}, \hat{R}_{\mathbf{j}}, \hat{B}_{\mathbf{j}}$ for $\mathbf{j} = 1, 2, \ldots, \mathbf{i} - 1$:*

$$\hat{U}_{\mathbf{j}} = U_{\mathbf{j}}, \quad \hat{R}_{\mathbf{j}} = R_{\mathbf{j}}, \tag{3.7}$$
$$\hat{B}_{\mathbf{j}} = R_{\mathbf{j}} (R_{\mathbf{i}_l} \cdots R_{\mathbf{i}_1}) H (R_{\mathbf{i}_l} \cdots R_{\mathbf{i}_1})^T R_{\mathrm{sib}(\mathbf{j})}^T, \tag{3.8}$$
$$\hat{D}_{\mathbf{j}} = U_{\mathbf{j}} R_{\mathbf{j}} (R_{\mathbf{i}_l} \cdots R_{\mathbf{i}_1}) H (R_{\mathbf{i}_l} \cdots R_{\mathbf{i}_1})^T R_{\mathbf{j}}^T U_{\mathbf{j}}^T, \quad (\mathbf{j}\text{: leaf}), \tag{3.9}$$

where $\mathbf{j} \to \mathbf{i}_l \to \cdots \to \mathbf{i}_1 \to \mathbf{i}$ is the path connecting $\mathbf{j}$ to $\mathbf{i}$.

Proof A construction procedure in [95] may be used to show this, but a more natural proof is by induction. That is, let $\mathcal{T}_{\mathbf{i}}$ be the subtree of $\mathcal{T}$ with root $\mathbf{i}$. The induction is done on the number of levels of $\mathcal{T}_{\mathbf{i}}$.

If $\mathcal{T}_{\mathbf{i}}$ has two levels,

$$U_{\mathbf{i}} H U_{\mathbf{i}}^T = \begin{pmatrix} U_{\mathbf{c}_1} R_{\mathbf{c}_1} \\ U_{\mathbf{c}_2} R_{\mathbf{c}_2} \end{pmatrix} H \begin{pmatrix} R_{\mathbf{c}_1}^T U_{\mathbf{c}_1}^T & R_{\mathbf{c}_2}^T U_{\mathbf{c}_2}^T \end{pmatrix} \tag{3.10}$$
$$= \begin{pmatrix} U_{\mathbf{c}_1}(R_{\mathbf{c}_1} H R_{\mathbf{c}_1}^T) U_{\mathbf{c}_1}^T & U_{\mathbf{c}_1}(R_{\mathbf{c}_1} H R_{\mathbf{c}_2}^T) U_{\mathbf{c}_2}^T \\ U_{\mathbf{c}_2}(R_{\mathbf{c}_2} H R_{\mathbf{c}_1}^T) U_{\mathbf{c}_1}^T & U_{\mathbf{c}_2}(R_{\mathbf{c}_2} H R_{\mathbf{c}_2}^T) U_{\mathbf{c}_2}^T \end{pmatrix}.$$

Then

$$\hat{B}_{\mathbf{c}_1} = R_{\mathbf{c}_1} H R_{\mathbf{c}_2}^T, \quad \hat{D}_{\mathbf{c}_1} = U_{\mathbf{c}_1}(R_{\mathbf{c}_1} H R_{\mathbf{c}_1}^T) U_{\mathbf{c}_1}^T, \quad \hat{D}_{\mathbf{c}_2} = U_{\mathbf{c}_2}(R_{\mathbf{c}_2} H R_{\mathbf{c}_2}^T) U_{\mathbf{c}_2}^T.$$

The results follow immediately.

Assume the results are true for $\mathcal{T}_{\mathbf{i}}$ with $2, 3, \ldots, l-1$ levels. We show they are also true for $\mathcal{T}_{\mathbf{i}}$ with l levels. In fact $\mathcal{T}_{\mathbf{c}_1}$ and $\mathcal{T}_{\mathbf{c}_2}$ has $l-1$ levels. By induction, $U_{\mathbf{c}_1}(R_{\mathbf{c}_1} H R_{\mathbf{c}_1}^T) U_{\mathbf{c}_1}^T$ is an HSS matrix with generators $D_{\mathbf{j}}, U_{\mathbf{j}}, \hat{R}_{\mathbf{j}}, \hat{B}_{\mathbf{j}}$ for $\mathbf{j} = 1, 2, \ldots, \mathbf{c}_1 - 1$, where

$$\hat{B}_{\mathbf{j}} = R_{\mathbf{j}}(R_{\mathbf{i}_l} \cdots R_{\mathbf{i}_2})(R_{\mathbf{c}_1} H R_{\mathbf{c}_1}^T)(R_{\mathbf{i}_l} \cdots R_{\mathbf{i}_2})^T R_{\mathrm{sib}(\mathbf{j})}^T$$
$$= R_{\mathbf{j}}(R_{\mathbf{i}_l} \cdots R_{\mathbf{i}_2} R_{\mathbf{c}_1}) H (R_{\mathbf{i}_l} \cdots R_{\mathbf{i}_2} R_{\mathbf{c}_1})^T R_{\mathrm{sib}(\mathbf{j})}^T.$$

Since $\mathbf{c}_1 = \mathbf{i}_1$ is the immediate descendent of $\mathbf{i}$ that is in the path from $\mathbf{j}$ to $\mathbf{i}$.

Similarly, apply induction to $U_{\mathbf{c}_2}(R_{\mathbf{c}_2} H R_{\mathbf{c}_2}^T) U_{\mathbf{c}_2}^T$, for $\mathbf{j} = \mathbf{c}_1 + 1, \mathbf{c}_1 + 2, \ldots, \mathbf{c}_2 - 1$. For the nonleaf node $\mathbf{j} = \mathbf{c}_1$, it obviously holds. To summarize, the results hold for all $\mathbf{j} = 1, 2, \ldots, \mathbf{i} - 1$. ■

Thus, by setting $\mathbf{i} \equiv \mathbf{k}$ in the Lemma, we can see that $U_{\mathbf{k}} H U_{\mathbf{k}}^T$ and A have the same U, R generators. They are usually said to have *common nested off-diagonal bases.* For such matrices, it is convenient to verify the following result.

Theorem 3.1.3 *Assume two conformably partitioned symmetric HSS matrices A and C have the same U, R generators, and the off-diagonal ranks of A and C are* *bounded by r. Then $A \pm C$ can be written as an HSS form with the same U, R generators as those of A and C, and the off-diagonal ranks of $A \pm C$ are bounded by r.*

Combining the results in the two lemmas, we have the following theorem for the fast HSS update in the dividing stage.

Theorem 3.1.4 *Use the same notation, and set $\mathbf{i} = \mathbf{k}$. The matrix $A - U_{\mathbf{k}} H U_{\mathbf{k}}^T$ has the same U, R generators as A, and its D, B generators can be obtained via the following updates:*

$$B_{\mathbf{j}} \leftarrow B_{\mathbf{j}} - R_{\mathbf{j}}(R_{\mathbf{i}_l} \cdots R_{\mathbf{i}_1}) H (R_{\mathbf{i}_l} \cdots R_{\mathbf{i}_1})^T R_{\mathrm{sib}(\mathbf{j})}^T, \tag{3.11}$$

$$D_{\mathbf{j}} \leftarrow D_{\mathbf{j}} - U_{\mathbf{j}} R_{\mathbf{j}}(R_{\mathbf{i}_l} \cdots R_{\mathbf{i}_1}) H (R_{\mathbf{i}_l} \cdots R_{\mathbf{i}_1})^T R_{\mathbf{j}}^T U_{\mathbf{j}}^T, \quad (\mathbf{j}\text{: } \textit{leaf}), \tag{3.12}$$

and the off-diagonal ranks of $A - U_{\mathbf{k}} H U_{\mathbf{k}}^T$ are bounded by r.

Note that this update involves the update of the generators associated with all the descendants of $\mathbf{i}$.

By setting $\mathbf{i}$ to be $\mathbf{c}_1$ and H to be I gives the HSS structure of $\tilde{D}_{\mathbf{c}_1}$. Similarly, set $\mathbf{i}$ to be $\mathbf{c}_2$ and H to be $B_{\mathbf{c}_1}^T B_{\mathbf{c}_1}$ to get the HSS structure of $\tilde{D}_{\mathbf{c}_2}$. The dividing procedure can then be recursively applied to $\tilde{D}_{\mathbf{c}_1}$ and $\tilde{D}_{\mathbf{c}_2}$. Theorem guarantees that the HSS structures are preserved throughout the recursive dividing procedure.

As a simple example, consider a symmetric HSS matrix A with 7 nodes in its HSS tree $\mathcal{T}$:

$$A \equiv D_7 = \begin{pmatrix} D_3 & U_3 B_3 U_6^T \\ U_6 B_3^T U_3^T & D_6 \end{pmatrix}, \quad U_3 = \begin{pmatrix} U_1 R_1 \\ U_2 R_2 \end{pmatrix}, \quad U_6 = \begin{pmatrix} U_4 R_4 \\ U_5 R_5 \end{pmatrix}$$
$$D_3 = \begin{pmatrix} D_1 & U_1 B_1 U_2^T \\ U_2 B_1^T U_1^T & D_2 \end{pmatrix}, \quad D_6 = \begin{pmatrix} D_4 & U_4 B_4 U_5^T \\ U_5 B_4^T U_4^T & D_5 \end{pmatrix}.$$

The dividing scheme works as

$$D_7 = \mathrm{diag}(\tilde{D}_3, \tilde{D}_6) + Z_7 Z_7^T,$$

where

$$Z_7 = \begin{pmatrix} U_3 \\ U_6 B_3^T \end{pmatrix} = \begin{pmatrix} U_1 R_1 \\ U_2 R_2 \\ U_4 R_4 B_3^T \\ U_5 R_5 B_3^T \end{pmatrix}.$$

and the generators of A are updated as follows to get those of $\tilde{D}_3$ and $\tilde{D}_6$:

- $B_1 \leftarrow B_1 - R_1 R_2^T$,
- $B_4 \leftarrow B_4 - R_4 B_3^T B_3 R_5^T$.
- $D_1 \leftarrow D_1 - U_1 R_1 R_1^T U_1^T$,
- $D_2 \leftarrow D_2 - U_2 R_2 R_2^T U_2^T$,
- $D_4 \leftarrow D_4 - U_4 R_4 B_3^T B_3 R_4^T U_4^T$,
- $D_5 \leftarrow D_5 - U_5 R_5 B_3^T B_3 R_5^T U_5^T$.

The two subproblems $\tilde{D}_3$ and $\tilde{D}_6$ are further divided via the following updates to the generators:

- $D_1 \leftarrow D_1 - U_1 U_1^T$,
- $D_2 \leftarrow D_2 - U_2 B_1^T B_1 U_2^T$,
- $D_4 \leftarrow D_4 - U_4 U_4^T$,
- $D_5 \leftarrow D_5 - U_5 B_4^T B_4 U_5^T$.

In general, to divide $D_{\mathbf{i}}$ we update all the B generators associated with the left nodes in $\mathcal{T}_{\mathbf{i}}$, and the D generators associated with the leaves. The update of the B, D generators can follow a top-down sweep, so as to reuse some computations. For example, once $B_{\mathbf{j}}$ for has been updated, then the update of $B_{\mathbf{c}}$ for a child $\mathbf{c}$ of $\mathbf{j}$ looks like

$$B_{\mathbf{c}} \leftarrow B_{\mathbf{c}} - R_{\mathbf{c}} R_{\mathbf{j}} (R_{\mathbf{i}_l} \cdots R_{\mathbf{i}_1}) H (R_{\mathbf{i}_l} \cdots R_{\mathbf{i}_1})^T R_{\mathbf{j}}^T R_{\mathrm{sib}(\mathbf{c})}^T,$$

where $(R_{\mathbf{i}_l} \cdots R_{\mathbf{i}_1})$ has already been computed. This thus can be performed recursively as follows. Initially for node $\mathbf{i}$, let

$$S_{\mathbf{i}} = I.$$

Then for $\mathbf{j}$, the update of $B_{\mathbf{j}}$ becomes

$$B_{\mathbf{j}} \leftarrow B_{\mathbf{j}} - R_{\mathbf{j}} S_{\mathrm{par}(\mathbf{j})} S_{\mathrm{par}(\mathbf{j})}^T R_{\mathrm{sib}(\mathbf{j})}^T.$$

Then let

$$S_{\mathbf{j}} = R_{\mathbf{j}} S_{\mathrm{par}(\mathbf{j})},$$

which is used for later updates. After all the B generators are updated, $S_{\mathbf{j}} = R_{\mathbf{j}} R_{\mathbf{k}_l} \cdots R_{\mathbf{k}_1}$ is already available. We further compute $U_{\mathbf{j}} S_{\mathbf{j}}$ and use it to update $D_{\mathbf{j}}$.

Here, in the dividing process, H is determined based on whether the above procedure is applied to the left or the right child branch of $\mathbf{i}$.

In addition, further computational savings are possible. Clearly, the D, B generators may need to be updated up to $l_{\max}$ times, where $l_{\max} = O(\log n)$ is the total number of levels in $\mathcal{T}$. As an improvement, we may accumulate the updates so as to save the intermediate multiplication costs for forming the updates. In practice, this may be skipped to simplify the algorithm, since the cost in the conquering stage later usually dominates the total cost (especially when r is very small).

3.1.3 Conquer

In the "conquering" stage, we compute the eigendecomposition of A from those of the subproblems. The rank-r update in (3.5) is split into r rank-1 updates. We start with the following case with a single rank-1 update:

$$\mathrm{diag}(\tilde{D}_{\mathbf{c}_1}, \tilde{D}_{\mathbf{c}_2}) + zz^T. \tag{3.13}$$

Just like in the standard DC, suppose we have computed the eigendecompositions $\tilde{D}_{\mathbf{c}_1} = \tilde{Q}_{\mathbf{c}_1} \tilde{\Lambda}_{\mathbf{c}_1} \tilde{Q}_{\mathbf{c}_1}^T$, $\tilde{D}_{\mathbf{c}_2} = \tilde{Q}_{\mathbf{c}_2} \tilde{\Lambda}_{\mathbf{c}_2} \tilde{Q}_{\mathbf{c}_2}^T$. Then

$$\mathrm{diag}(\tilde{D}_{\mathbf{c}_1}, \tilde{D}_{\mathbf{c}_2}) + zz^T = \mathrm{diag}(\tilde{Q}_{\mathbf{c}_1}, \tilde{Q}_{\mathbf{c}_2})(\tilde{\Lambda} + vv^T)\,\mathrm{diag}(\tilde{Q}_{\mathbf{c}_1}^T, \tilde{Q}_{\mathbf{c}_2}^T), \tag{3.14}$$

where

$$\tilde{\Lambda} = \mathrm{diag}(\tilde{\Lambda}_{\mathbf{c}_1}, \tilde{\Lambda}_{\mathbf{c}_2}) \equiv \mathrm{diag}(\tilde{\lambda}_j, j = 1, 2, \ldots), \quad v = \mathrm{diag}(\tilde{Q}^T_{\mathbf{c}_1}, \tilde{Q}^T_{\mathbf{c}_2})z. \tag{3.15}$$

Here, we can assume all $\tilde{\lambda}_j$'s are distinct, and v has no zero entry. Otherwise, the deflation strategy in Section 3.1.3 is applied. In the following, we discuss how to quickly find the eigendecomposition of (3.13) with the aid of FMM.

We keep each part of this subsection compact, since much of the technical details can be generalized from [19, 39] (although the actual algorithm design and implementation are far less trivial). We include only essential descriptions to introduce necessary notation and to sketch the basic ideas. Some pseudocodes will be included to assist in the understanding.

FMM in one dimension

FMM in one dimension will be used at multiple places in our algorithm. Here, we only briefly mention its basic idea. The reader is referred to [6, 14, 15, 35] for more details.

Suppose we wish to evaluate the following function at multiple points λ:

$$\Phi(\lambda) = \sum_{j=1}^{N} \alpha_j \phi(\lambda - \tilde{\lambda}_j), \tag{3.16}$$

where $\{\tilde{\lambda}_j\}_{j=1}^N$ are given real points, $\{\alpha_j\}_{j=1}^N$ are constants, and $\phi(x)$ is a specific kernel function of interest. In our case, $\phi(x)$ is either $1/x$, $\log(x)$, or $1/x^2$. FMM is designed to quickly evaluate $\Phi(\lambda)$ at M points $\{\lambda_i\}_{i=1}^M$ without using the dense matrix-vector multiplication $K\alpha$, where $K = (\phi(\lambda_i - \tilde{\lambda}_j))_{M\times N}$.

The FMM implementation we use is based on [14], where explicit accuracy and stability estimates are given. We briefly describe the results here. Suppose (a, b) and (c, d) are two well-separated intervals and $\lambda_i \in (a, b), i = 1, \ldots, M,\ \tilde{\lambda}_j \in (c, d), j = 1, \ldots, N$. Compute a truncated Taylor series expansion of ϕ:

$$\phi(\lambda - \tilde{\lambda}) \approx \sum_{k=1}^{p} f_k(\lambda) g_k(\tilde{\lambda}), \tag{3.17}$$

where a proper scaling is applied to f_k and g_k. The relative approximation error is [14]

$$\epsilon = \frac{1+\eta}{1-\eta}\eta^p, \tag{3.18}$$

where $\eta \in (0,1)$ depends on the separation between (a,b) and (c,d). Thus, p only needs to be $O(\log \tau)$ to reach a desired accuracy τ. Then, this enables us to write a low-rank approximation

$$K = (\phi(\lambda_i - \tilde{\lambda}_j))_{M\times N} \approx \underset{M\times p}{\hat{U}} \cdot \underset{p\times p}{\hat{C}} \cdot \underset{p\times N}{\hat{V}^T},$$

where the elementwise relative approximation error is given. Furthermore, the proper scaling of the Taylor series expansion guarantees that the entries of $\hat{U}$ and $\hat{V}$ have magnitudes bounded by 1, and the entries of $\hat{C}$ have magnitudes roughly proportional to those of K [14, Section 6.2]. This enables us to stably evaluate $K\alpha$ to a desired accuracy with complexity $O(M+N)$ instead of $O(MN)$.

When λ_i and $\tilde{\lambda}_j$ come from the same set of points, then the process is done hierarchically as in the standard FMM so as to reach the overall linear complexity. In our implementation, we make the separation parameter $\eta \le \frac{2}{3}$ and the accuracy τ to be around 10^{-10} or even smaller.

Note that when FMM is used in our DC algorithm, it implicitly approximates the intermediate matrices. For example, an elementwise relative error ϵ is introduced into the intermediate eigenmatrices. Such an error may be propagated to later computations. Due to the hierarchical DC scheme, it is expected that the error may be magnified by only up to about $\log n$ times, similar to the approximation error results in [4,34]. Such error propagations are thus well controlled, and in practice, the accuracy of the eigenvalues is consistent with the tolerance. We can similarly understand the behaviors of the numerical errors in the FMM matrix-vector multiplication, just like the stability analysis for a hierarchical matrix factorization in [89].

Computing the eigenvalues by solving the secular equation

As in the tridiagonal DC scheme, the eigenvalues λ are the roots of the secular equation

$$f(\lambda) = 1 + \sum_{j=1}^{n} \frac{v_j^2}{\tilde{\lambda}_j - \lambda} = 0, \tag{3.19}$$

which can be solved with Newton's method. To ensure the quick and stable solution of (3.19), we follow the modified Newton's method in [22], which is based on the Middle Way in [64]. This modified Newton's method involves the evaluation of functions of the forms $\varphi(\lambda) = \sum_{j=1}^{n} \frac{v_j^2}{\tilde{\lambda}_j - \lambda}$ and $\varphi'(\lambda)$ for multiple λ. This can be accelerated by FMM with $\phi(x) = 1/x$ or $1/x^2$ in (3.16).

Just as mentioned in [22], two or three Newton iterations are sufficient to reach the machine precision. This strategy works for all the roots of the secular equation except the largest one, for which we follow [39] and use basic rational interpolation with several safeguards for stability based on the algorithm in [13].

Computing the eigenvectors stably

As has been extensively studied [21, 24, 78], the computation of the eigenvectors via the simple formula $q_j = (\tilde{\Lambda} - \lambda_j I)^{-1} v$ can have stability issues. In particular, if $|\lambda_i - \lambda_j|$ is small for two eigenvalues λ_i and λ_j, the corresponding eigenvectors q_i and q_j may be far from orthogonal [24]. A stable computational strategy [39] is to solve for the eigenvectors of a slightly perturbed problem $\tilde{\Lambda} + \hat{v}\hat{v}^T$, which has the exact eigenvalues λ_j. The vector $\hat{v} = (\hat{v}_i)_{k=1}^n$ is computed based on Löwner's formula [22]:

$$\hat{v}_i = \sqrt{\frac{\prod_{j=1}^{i-1}(\tilde{\lambda}_i - \lambda_j) \prod_{j=i}^{n}(\lambda_j - \tilde{\lambda}_i)}{\prod_{j=1}^{i-1}(\tilde{\lambda}_i - \tilde{\lambda}_j) \prod_{j=i+1}^{n}(\tilde{\lambda}_j - \tilde{\lambda}_i)}}, \tag{3.20}$$

where the eigenvalues are ordered from the largest to the smallest. The vector $\hat{v}$ can be quickly evaluated with FMM applied to $\log \hat{v}_i$ [39]. That is, set $\phi(x) = \log(x)$ in (3.16).

The eigenvectors associated with all the eigenvalues λ_j can be assembled into a matrix $\hat{Q} \equiv \left(\frac{\hat{v}_i}{\tilde{\lambda}_i - \lambda_j}\right)_{i,j}$. While we do not explicitly form this matrix, we still need to normalize its columns to obtain orthonormal eigenvectors and to ensure the stability of later calculations. Let s_j be the inverse of the norm of column j of $\hat{Q}$. It is used to scale that column as in

$$Q_{\mathbf{i}}^{(1)} = \left(\frac{\hat{v}_i s_j}{\tilde{\lambda}_i - \lambda_j}\right)_{i,j}, \quad \text{with} \quad s_j = \left(\sum_{i=1}^{n} \frac{v_i^2}{(\tilde{\lambda}_i - \lambda_j)^2}\right)^{-1/2}, \tag{3.21}$$

where the superscript in $Q_{\mathbf{i}}^{(1)}$ is used to indicate that the result is from a single rank-1 update (3.13). Once again, the computation of s_j can be accelerated by FMM, with $\phi(x) = 1/x^2$ in (3.16). Note that $\hat{Q}$ is now converted into the orthogonal Cauchy-like matrix $Q_{\mathbf{i}}^{(1)}$ (a Cauchy-like matrix is a matrix whose (i, j) entry looks like $\frac{\alpha_i \beta_j}{d_i - f_j}$ for four vectors α, β, d, f).

Rank-r updated eigendecomposition

The above process needs to be repeated r times for the rank-r update in (3.5). We summarize the process in the following lemma and skip the details.

Theorem 3.1.5 *Suppose $\tilde{D}_{\mathbf{c}_1} = \tilde{Q}_{\mathbf{c}_1} \tilde{\Lambda}_{\mathbf{c}_1} \tilde{Q}_{\mathbf{c}_1}^T$ and $\tilde{D}_{\mathbf{c}_2} = \tilde{Q}_{\mathbf{c}_2} \tilde{\Lambda}_{\mathbf{c}_2} \tilde{Q}_{\mathbf{c}_2}^T$ are the eigendecompositions of $\tilde{D}_{\mathbf{c}_1}$ and $\tilde{D}_{\mathbf{c}_2}$ in (3.5), respectively. Let*

$$Z_{\mathbf{i}} = (z^{(1)}, \ldots, z^{(r)}), \quad Q^{(0)} = \operatorname{diag}(\tilde{Q}_{\mathbf{c}_1}, \tilde{Q}_{\mathbf{c}_2}), \quad v^{(0)} = (Q^{(0)})^T z, \quad \lambda_j^{(0)} = \tilde{\lambda}_j.$$

Suppose the eigendecomposition of $\operatorname{diag}(\lambda_j^{(i-1)}|_{j=1}^n) + v^{(i)}(v^{(i)})^T$ *is*

$$\operatorname{diag}(\lambda_j^{(i-1)}|_{j=1}^n) + v^{(i)}(v^{(i)})^T = Q_{\mathbf{i}}^{(i)} \operatorname{diag}(\lambda_j^{(i)}|_{j=1}^n)(Q_{\mathbf{i}}^{(i)})^T,$$

where $Q_{\mathbf{i}}^{(i)}$ is in a Cauchy-like form and $v^{(i)} = (Q_{\mathbf{i}}^{(i-1)})^T z^{(i)}$. Then the eigendecomposition of $D_{\mathbf{i}}$ in (3.5) is

$$D_{\mathbf{i}} = (Q_{\mathbf{i}}^{(0)} Q_{\mathbf{i}}) \operatorname{diag}(\lambda_j^{(r)}|_{j=1}^n)(Q_{\mathbf{i}}^{(0)} Q_{\mathbf{i}})^T,$$

where

$$Q_{\mathbf{i}} = Q_{\mathbf{i}}^{(1)} \cdots Q_{\mathbf{i}}^{(r)}. \tag{3.22}$$

That is, $\lambda_j^{(r)}|_{j=1}^n$ are the eigenvalues of $D_{\mathbf{i}}$ in (3.5) and $Q_{\mathbf{i}}^{(0)}Q_{\mathbf{i}}$ is the eigenmatrix of $D_{\mathbf{i}}$. For completeness, if $\mathbf{i}$ is a leaf node of the HSS tree, we set $Q_{\mathbf{i}}^{(0)} = I$ and compute $Q_{\mathbf{i}}$ directly via the eigendecomposition of the diagonal block $D_{\mathbf{i}}$.

Application of the eigenmatrix to vectors and structure of the eigenmatrix

Note that we do not form the eigenmatrix $Q_{\mathbf{i}}^{(0)}(Q_{\mathbf{i}}^{(1)} \cdots Q_{\mathbf{i}}^{(r)})$ of $D_{\mathbf{i}}$ or the eigenmatrix Q of A explicitly. In practical applications, the eigenvectors of A are often used under the following circumstance: applications of the eigenmatrix or its transpose to vectors. In fact, such a process is already needed in the DC process for computing v in (3.15). Thus, we illustrate this as part of the eigendecomposition.

For an individual matrix $Q_{\mathbf{i}}^{(1)}$ of the form (3.21), to multiply $(Q_{\mathbf{i}}^{(1)})^T$ and a vector z, we have

$$\left((Q_{\mathbf{i}}^{(1)})^T z\right)_j = s_j \sum_{i=1}^{n} \frac{\hat{v}_i z_i}{\tilde{\lambda}_i - \lambda_j}. \tag{3.23}$$

Similarly to [19, 39], this can be accelerated by FMM with $\phi(x) = 1/x$ in (3.16). To apply $Q_{\mathbf{i}}$ to a vector, we just need to repeat this r times.

The overall strategy for applying Q or Q^T to a vector z is basically the one in [19, 22]. For our case, this can be done with the aid of the HSS tree $\mathcal{T}$. More specifically, associate $Q_{\mathbf{i}}$ in (3.22) with each node $\mathbf{i}$ of $\mathcal{T}$. Then we use a multilevel procedure to compute the eigenmatrix-vector product.

For convenience, Algorithm 2 shows how to apply Q^T to z, as needed in forming v in (3.15). The multiplication of Q and z can be performed similarly, and can be used if we need to extract any specific column of Q.

Clearly, the data-sparse structure of Q defined by $Q_{\mathbf{1}}, \ldots, Q_{\mathbf{k}}$ in their Cauchy-like forms is very useful for the fast application of Q or Q^T to a vector. On the other hand, we may also understand the data sparsity of Q based on its off-diagonal rank structure. It can be shown that the eigenmatrix of $\tilde{\Lambda} + \hat{v}\hat{v}^T$ has off-diagonal numerical ranks at most $O(\log n)$ for a given tolerance. Thus, the off-diagonal numerical ranks

Algorithm 2 Application of Q^T to a vector, where Q is the eigenmatrix of A

```
 1: procedure eigmv($Q_{\mathbf{1}}, \dots, Q_{\mathbf{k}}, z$)  Output: $Q^T z$, where $Q$ is represented by $Q_{\mathbf{1}}, \dots, Q_{\mathbf{k}}$
 2:     Partition $z$ into pieces $z_{\mathbf{i}}$ following the sizes of $D_{\mathbf{i}}$ for all leaves $\mathbf{i}$
 3:     for $\mathbf{i} = 1, \dots, \mathbf{k}$ do                              ▷ $\mathbf{k}$: root of $\mathcal{T}$
 4:         if $\mathbf{i}$ is a nonleaf node then                 ▷ $\mathbf{c}_1, \mathbf{c}_2$: children of $\mathbf{i}$
 5:             $z_{\mathbf{i}} \leftarrow \begin{pmatrix} z_{\mathbf{c}_1} \\ z_{\mathbf{c}_2} \end{pmatrix}$
 6:         end if
 7:         for $i = 1, 2, \dots, r$ do                 ▷ $r$: column size of $Z_{\mathbf{i}}$ in (3.5)
 8:             $z_{\mathbf{i}} \leftarrow (Q_{\mathbf{i}}^{(i)})^T z_{\mathbf{i}}$ (fast evaluation via FMM)          ▷ As in (3.23)
 9:         end for
10:     end for
11:     Output $z_{\mathbf{k}}$                                        ▷ $z_{\mathbf{k}} = Q^T z$
12: end procedure
```

of Q are at most $O(r \log^2 n)$. Since this rank structure of Q is not actually used in our algorithms, we omit the details.

Deflation

If the vector v has a zero entry, or if $\tilde{\Lambda}$ has two equal diagonal entries, deflation strategies can be applied. This is already shown in [24, 39]. For example, if $v_j = 0$, then $\tilde{\Lambda} + vv^T$ has an eigenvalue

$$\lambda_j = \tilde{\lambda}_j.$$

If $\tilde{\Lambda}$ has two (or more) identical diagonal entries $\tilde{\lambda}_i = \tilde{\lambda}_j$, then a Householder transformation can be used to zero out v_j so as to convert into the previous case. (In these cases, the eigenmatrix of $\tilde{\Lambda} + vv^T$ is then block diagonal and may involve Cauchy-like or Householder diagonal blocks.) A similarly strategy can be applied if v_j is small or if the difference between $\tilde{\lambda}_i$ and $\tilde{\lambda}_j$ is small, and the detailed perturbation analysis is provided in [24, 39]. This step is standard but important for the efficiency of the algorithm.

Rank structure of the eigenmatrix

According to the next section, we see that the cost of applying Q or Q^T to a vector is $O(rn\log n)$. It is worth pointing out that Q is also rank structured, although the off-diagonal numerical ranks are slightly higher. To see this, we first state a lemma, which is obvious based on FMM (see, e.g. [6, 14, 79]).

Theorem 3.1.6 *Assume all the eigenvalues $\tilde{\lambda}_i$ and λ_j are distinct, then the Cauchy-like matrix has off-diagonal numerical ranks $O(\log n)$ for a given tolerance.*

However, unlike the tridiagonal DC, here repeated eigenvalues may arise (theoretically). For this case, the eigenmatrix can be represented by a block diagonal form with the block structure conforming exactly to that of HSS approximation with the eigenvector structure at each node equal to a concatenation of two matrices of the following types:

- Householder forms as mentioned for the repeated eigenvalues;
- Cauchy-like forms similar for remaining distinct eigenvalues.

Obviously, a Householder matrix has maximum off-diagonal rank 1. Thus,

Proposition 3.1.1 *The eigenmatrix of $\tilde{\Lambda} + \hat{v}\hat{v}^T$ has off-diagonal numerical ranks $O(\log k)$ for a given tolerance, where k is the number of distinct diagonal entries of $\tilde{\Lambda}$.*

In practice, we may have clustered eigenvalues instead of repeated ones. While this does not change the theoretical rank structure (as in tridiagonal DC), it may cause difficulties to the actual computations. The detailed study is beyond the focus of this paper and will be given in future work. But it has been commonly observed [19,22,24,39] that clustered eigenvalues pose a problem for classical divide-and-conquer eigensolvers since this can cause the eigenvectors to lose orthogonality and the evaluation of the secular equation diverges. One thing we would like to point out is, even though the Cauchy-like form may be very ill conditioned in this case, hierarchical rank structured methods are still very reliable due to a significant stability advantage over standard methods [89]. That is, the numerical errors only propagate along the tree levelwise or by at most $O(\log n)$ times. We would like to mention that this is also related to the following known result, which indicates that symmetric Toeplitz problems are nice applications of our superfast DC method.

Theorem 3.1.7 *[61, 71] Let T be a Toeplitz matrix, and C be a Cauchy-like matrix resulting from the transformation of T into Fourier space through the use of displacement structures [31, 37, 50, 63, 70]. Then the off-diagonal numerical ranks of C are $O(\log n)$ for a given tolerance.*

With these results, all the matrices $Q_{\mathbf{i}}^{(1)}$, ..., $Q_{\mathbf{i}}^{(r)}$ have off-diagonal numerical ranks at most $O(\log n)$. A straightforward multiplication indicates that $Q_{\mathbf{i}}$ has off-diagonal numerical ranks at most $O(r \log n)$, which is the contribution to the off-diagonal ranks of Q from the current dividing level. The recursive dividing procedure is performed for $O(\log n)$ levels, so we need to accumulate such rank contributions from all the $O(\log n)$ levels. This yields the following result on the rank structure of Q.

Theorem 3.1.8 *The off-diagonal numerical ranks of Q are at most $O(r \log^2 n)$ for a given tolerance.*

Therefore, together with the Cauchy-like/Householder structures is a natural way to multiply the eigenmatrix and a vector. We usually do not convert Q into a rank structured matrix, since it would cost more. Lastly, the rank bounds in this subsection will also be useful For without the rank-property, a complexity below $O(n^2)$ would not be able to be achieved, since the eigenmatrix-vector products would then not be able to be accelerated.

Table 3.1.
Example (KMS Toeplitz matrix): Complexity ξ of NEW for finding all the eigenvalues (as compared with XXC14), complexity $\tilde{\xi}$ of NEW for applying the eigenmatrix to a vector, and storage σ of NEW for the eigenmatrix.

	n	160	320	640	1280	2560	5120	10240
XXC14	ξ	$3.17e08$	$1.19e09$	$4.72e09$	$1.82e10$	$7.14e10$	$2.82e11$	$1.12e12$
NEW	ξ	$1.55e08$	$5.45e08$	$1.70e09$	$4.86e09$	$1.32e10$	$3.44e10$	$8.70e10$
	$\tilde{\xi}$	$3.40e05$	$1.26e06$	$4.53e06$	$1.36e07$	$3.81e07$	$1.01e08$	$2.61e08$
	σ	$3.84e03$	$1.02e04$	$2.56e04$	$6.14e04$	$1.43e05$	$3.28e05$	$7.37e05$

3.1.4 Numerical results

- `NEW`: our superfast DC eigensolver;
- `XXC14`: the HSS eigensolver in [90];
- λ_i: the eigenvalues of A (here, the results from the Matlab function `eig` are used as the "exact" eigenvalues);
- $\hat{\lambda}_i$: the numerical eigenvalues;
- $\hat{Q}$: the numerical eigenmatrix with column $\hat{q}_i$ being the numerical eigenvector associated with $\hat{\lambda}_i$;
- $\gamma = \frac{\max_i \|A\hat{q}_i - \hat{\lambda}_i \hat{q}_i\|_2}{n\|A\|_2}$: the residual, as used in [39];
- $\theta = \frac{\max_i \|\hat{Q}^T\hat{q}_i - e_i\|_2}{n}$: the loss of orthogonality, as used in [39];
- $e = \frac{\sqrt{\Sigma_{i=1}^n (\lambda_i - \hat{\lambda}_i)^2}}{n\sqrt{\Sigma_{i=1}^n \lambda_i^2}}$: the relative error;
- $\xi, \tilde{\xi}, \sigma$: complexity measurements as in [88]

The HSS block sizes are chosen following the strategies in common HSS practices. A tolerance is used in the HSS approximation and FMM (if applicable) so that both NEW and XXC14 reach accuracies e around 10^{-10}. A smaller tolerance is also tested for NEW to reach higher or even the machine accuracy.

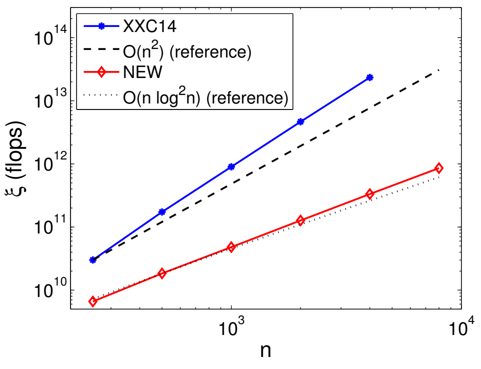
We first consider a banded symmetric matrix A with full bandwidth 10 and with its nonzero entries uniformly distributed in $[-1, 1]$.

We test A with its size n varying from 250 to 8000. In Table 3.2, we show the complexity ξ of NEW and XXC14 to reach similar accuracies in the eigenvalues. Clearly, the asymptotic complexity scales like $O(n \log^2 n)$ in NEW and $O(n^2)$ in XXC14, as also illustrated in Figure 3.1(i). The reference lines for $O(n \log^2 n)$ and $O(n^2)$ are also shown. For $n = 4000$, NEW is already about 70 times faster than XXC14.

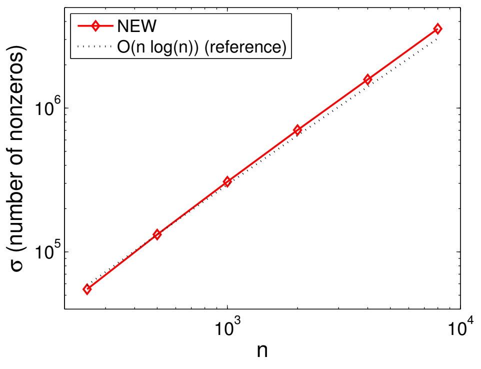
NEW further gives a structured eigenmatrix Q, which can be applied quickly to a vector. See Table 3.2 for its cost $\tilde{\xi}$ and the storage σ. The storage is also plotted in Figure 3.1(ii), and scales like $O(n \log n)$. On the other hand, the eigenmatrix is not available from XXC14.

Table 3.2.
Example 3.1.4 (banded random matrix): Complexity ξ of NEW for finding all the eigenvalues (as compared with XXC14), complexity $\tilde{\xi}$ of NEW for applying the eigenmatrix to a vector, and storage σ of NEW for the eigenmatrix.

	n	250	500	1000	2000	4000	8000
XXC14	ξ	$3.01e10$	$1.73e11$	$9.01e11$	$4.66e12$	$2.34e13$	Failed
NEW	ξ	$6.62e09$	$1.84e10$	$4.80e10$	$1.27e11$	$3.33e11$	$8.56e11$
	$\tilde{\xi}$	$1.25e07$	$3.43e07$	$9.05e07$	$2.32e08$	$5.90e08$	$1.48e09$
	σ	$5.50e04$	$1.32e05$	$3.08e05$	$7.04e05$	$1.58e06$	$3.56e06$



(i) Eigenvalue solution cost ξ



(ii) Structured eigenmatrix storage σ

Fig. 3.1. Example 3.1.4 (banded random matrix): Complexity ξ of NEW and XXC14 for finding all the eigenvalues, and storage σ of NEW for the eigenmatrix.

The accuracies are shown in Table 3.3, following the measurements in [39]. Both methods reach similar accuracies in the eigenvalues. Since NEW also produces the eigenvectors, we report the residual γ and the orthogonality measurement θ. In particular, θ for NEW reaches nearly machine accuracy.

Table 3.3.
Example 3.1.4 (banded random matrix): Accuracy (error e, residual γ, and loss of orthogonality θ) of the methods.

	n	250	500	1000	2000	4000
XXC14	e	$2.28e-10$	$1.43e-10$	$5.59e-11$	$3.31e-11$	$2.27e-11$
NEW	e	$2.96e-11$	$3.15e-11$	$9.02e-10$	$8.31e-11$	$8.32e-11$
	γ	$4.39e-11$	$3.23e-11$	$1.42e-10$	$8.47e-11$	$6.48e-11$
	θ	$4.29e-16$	$7.39e-16$	$1.53e-15$	$3.99e-15$	$8.73e-15$

Seconly, we consider the Kac-Murdock-Szego (KMS) Toeplitz matrix A as in [82], with its entries given by

$$A_{ij} = \rho^{|i-j|}, \quad \rho = 0.5.$$

The matrix A is known to be very ill conditioned. The ill-conditioning implies that there are both very large and small eigenvalues, which we show that we are able to capture to near machine precision by testing this matrix with multiple tolerances. A has the same eigenvalues as the Cauchy-like matrix C in, and our tests are done on C.

The maximum off-diagonal numerical rank of C grows with n as $O(\log n)$. We test C with sizes n ranging from 160 to 10240, and show the complexity ξ of `NEW` and `XXC14` to reach similar accuracies in the eigenvalues. The performance results are given. `NEW` takes less work than `XXC14` for all the cases. For $n = 10240$, `NEW` is over 12 times more efficient.

`NEW` further gives a structured eigenmatrix Q, which can be applied quickly to a vector. The eigenmatrix is not available from `XXC14`.

We have also compared `NEW` with the Matlab built-in `eig` function, which is highly optimized. Our algorithm is initially slower for smaller n, but scales much better. For $n = 2560, 5120, 10240$, the runtimes of `NEW` are 12.3, 40.0, 80.9 seconds, respectively (on a MacBook Pro with an Intel Core i7 CPU and 8GB memory), and those of `eig` are 5.1, 36.6, 270.0 seconds, respectively. Clearly, even if our code is far less optimized and the Matlab runtime is pessimistic for non-built-in routines, `NEW` already shows significant advantages for larger n.

We indicate the cost and storage are consistent with the theoretical prediction. We also note the comparison with the MATLAB `eig` function. For small matrices, `eig` is faster in terms of computation time. This is because of the large constant in the $O(r^2 n \log n) + O(rn \log^2 n)$ in our algorithm, and also because of optimization of the built-in MATLAB `eig` function. Still, around $n = 500$, there is a crossover point and our algorithm is faster for KMS matrices than the $O(n^3)$ complexity `eig` function. Very similar results hold for other HSS matrices.

The accuracies are shown. Both methods reach similar accuracies in the eigenvalues. Since `NEW` also produces the eigenvectors, we report the residual γ and the orthogonality measurement θ. In particular, θ for `NEW` reaches nearly machine precision. he accuracy measure γ used is the square of the residual normalized by the spectral norm of the matrix. This is done for two reasons. The first is historical, as this is the measure used in [39], which contains the algorithm we generalize, showing that we get similarly satisfactory results. The second reason is that is allows one to see how many digits of accuracy are preserved for a given HSS approximation.

Table 3.4.
Example ((KMS Toeplitz matrix): Accuracy (error e, residual γ, and loss of orthogonality θ) of the methods when the tolerance in the off-diagonal compression and FMM is set to be around 10^{-10}.

	n	160	320	640	1280	2560
`XXC14`	e	$2.40e-10$	$1.02e-10$	$5.80e-11$	$4.39e-11$	$3.84e-11$
`NEW`	e	$1.00e-09$	$1.07e-10$	$1.47e-10$	$9.32e-11$	$8.45e-11$
	γ	$3.49e-09$	$1.49e-09$	$7.38e-10$	$2.53e-10$	$9.99e-11$
	θ	$1.79e-16$	$3.69e-16$	$7.94e-16$	$6.56e-16$	$8.53e-16$

We would like to point out that, with a smaller tolerance, the residual and error in `NEW` can reach nearly machine precision too, as shown below. The corresponding cost of `NEW` is higher than with the 10^{-10} tolerance, but still scales like $O(n \log^3 n)$.

As mentioned at the beginning of this section, the residual measurement we use follows [39] and is not the regular one, so as to show that our structured DC eigensolver can reach desired accuracies, and can also reach machine precision. We have also checked the regular accuracy measurements. For the tests, the regular errors $|\lambda_i - \hat{\lambda}_i|$ are in the magnitudes around $10^{-19} \sim 10^{-16}$, mostly $10^{-19} \sim 10^{-17}$. (The errors are consistent with the bound.) The regular residuals $\|A\hat{q}_i - \hat{\lambda}_i\hat{q}_i\|_2$ are around $10^{-11} \sim$

Table 3.5.
Example (KMS Toeplitz matrix): Accuracy (error e, residual γ, and loss of orthogonality θ) of NEW when the tolerance in the off-diagonal compression and FMM is set to be around 10^{-15}.

	n	160	320	640	1280	2560
NEW	e	$9.64e-16$	$1.01e-15$	$1.27e-15$	$1.07e-15$	$1.31e-15$
	γ	$4.14e-15$	$4.40e-15$	$6.69e-15$	$7.62e-15$	$6.26e-15$
	θ	$4.25e-16$	$5.33e-16$	$7.24e-16$	$9.37e-16$	$7.18e-16$

10^{-10}. We have also computed the gaps $\hat{g}_i = \min_{j\neq i}|\hat{\lambda}_i - \lambda_j|$, which are around $10^{-6} \sim 10^{-3}$. It is known that if $\hat{\lambda}_i$ is the Rayleigh quotient of A and $\hat{q}_i$, then $|\lambda_i - \hat{\lambda}_i| \leq \|A\hat{q}_i - \hat{\lambda}_i\hat{q}_i\|_2^2/\hat{g}_i$ [22]. Here, our results are observed to roughly follow such a relationship.

Next we look at a symmetric Toeplitz matrix T with its first column given by

$$t_{0:n-1} = \text{randn}(n,1). \tag{3.24}$$

Here we show the performance of our algorithm for symmetric Toeplitz matrices with increasing sizes to illustrate scaling properties. Note that for large n the scaling is nearly linear. This near linear scaling appears even for moderately size matrices for matrices with HSS rank small compared to the matrix size. This is illustrated below which plots Example 2 and Example 3, which has lower HSS ranks, side-by-side.

For our final example, we consider a matrix we denote as B chosen according to formula

$$A_{i,j} = \sqrt{|x_i^{(n)} - x_j^{(n)}|}, \tag{3.25}$$

with the points $x_i^{(n)} = \cos(\pi(2i+1)/2n)$, the zeros of the nth Chebyshev polynomial. This is a symmetric HSS matrix. Its HSS rank r does grow as the matrix size n grows, but as in Example 2, the growth is moderate. Thus we are still able to exhibit

Table 3.6.
Flops and ratio of Flops for finding all eigenvalues of a random Toeplitz matrix of size n.

n	1280	2560	5120	10240	20480
HSS rank (r)	66	71	76	82	88
Flops	4.38e10	1.29e11	3.68e11	1.05e12	2.94e12
ratio n_2/n_1		2	2	2	2
ratio $O(n\log^3 n)$		2.64	2.58	2.53	2.49
ratio NEW		2.93	2.87	2.86	2.79
ratio $O(n^2)$		4.00	4.00	4.00	4.00

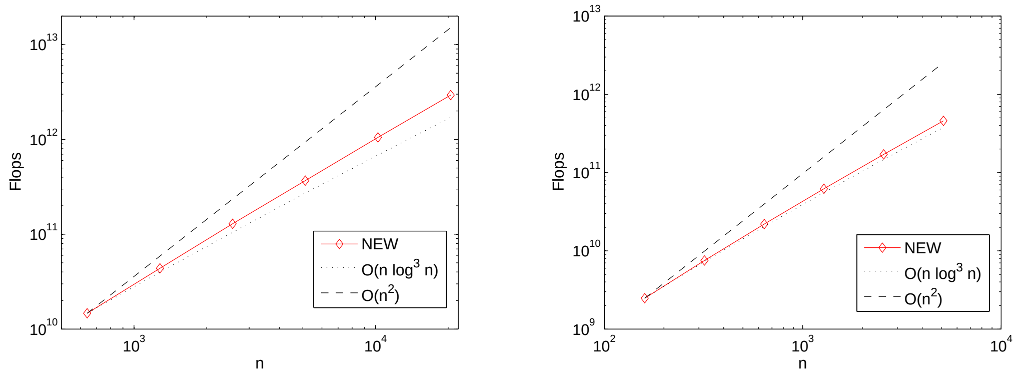


Fig. 3.2. Left: complexity versus matrix size for finding all eigenvalues of a random Toeplitz matrix of size n. Right: complexity versus matrix size for finding all eigenvalues of a matrix of type B with size n. In both cases note that $\tau = 1e - 6$.

near linear scaling. Complexity results for this examples are plotted above against appropriate reference lines.

Table 3.7.
Flops and ratio of Flops for finding all eigenvalues of a rmatrix of type B with size n.

n	320	640	1280	2560	5120
HSS rank (r)	10	10	11	12	13
Flops	7.54e09	2.20e10	6.21e10	1.71e11	4.59e11
ratio n_2/n_1		2	2	2	2
ratio $O(n\log^3 n)$		2.81	2.72	2.64	2.58
ratio NEW		2.92	2.82	2.75	2.69
ratio $O(n^2)$		4.00	4.00	4.00	4.00

3.2 Multi-rank update

While the above algorithm is certainly very impressive, it has some fundamental issues that limit the types of problems it can be used for. First of all, due to the conditioning of the secular equation (see figure), we can see that for clustered eigenvalues poor conditioning will be amplified if updates are repeated consecutively. Thus we need to find a stabler way to do this process to allow us to consider more pathological problems.

$$\kappa = \left(\frac{2}{u_i - l_i}\right)\left(\sum_i \frac{2v_i^2}{(u_i-l_i)^2}\right)\left(1+\sum_i \frac{2v_i^2}{u_i-l_i}\right)^{-1} \tag{3.26}$$

Secondly, this approach is not very useful in the context of parallel computing. It has poor spacial and temporal locality, and is overly reliant on BLAS 1 and BLAS 2 operations. thus we need to find a way to compute all rank-one updates simultaneously. This section presents a possible approach to do this. It is still ongoing work.

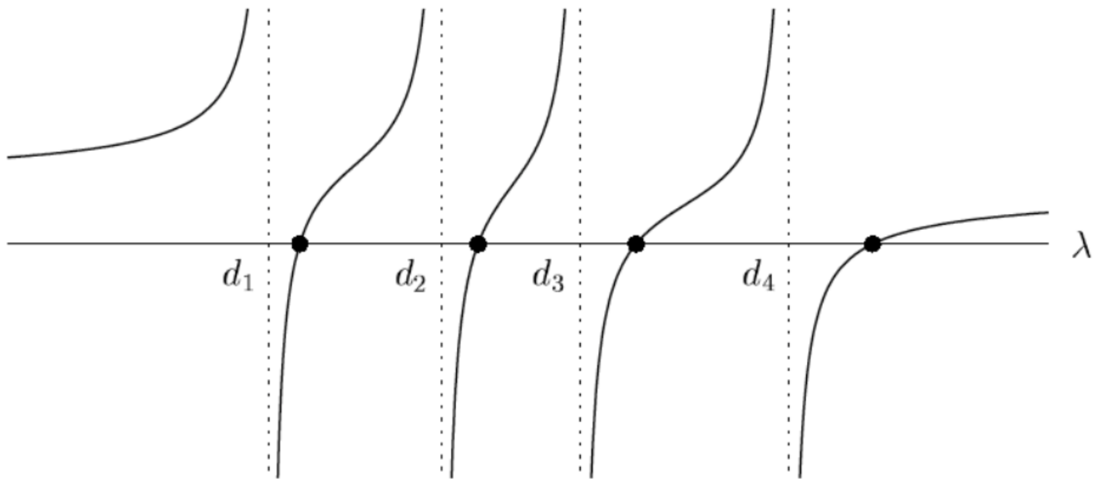


Fig. 3.3. The secular equation for the single-rank update

3.2.1 Eigenvalue computation

For a fixed eigenvalue λ, it is straightforward to identify an exact reduced basis of dimension r with an unknown vector of coefficients $w \in \mathbb{R}^r$,

$$q(\lambda) = (\lambda I - \Sigma)^{-1} U w. \tag{3.27}$$

We can split this basis into a small number of (nearly) singular components in $\lambda \in [\lambda_{\min}, \lambda_{\max}]$ and a smooth component that is amenable to interpolation from a small number of interpolation points $\hat{\lambda}_i$. We then project the eigenvalue problem into this reduced basis. It is convenient to orthogonalize the complete basis, so the resulting problem is a generalized eigenvalue problem.

We use a Cauchy kernel interpolation that is defined by three parameters $\sigma_{\min}, \sigma_{\max}$, and s. s is the number of interpolation points. $\hat{\lambda}_i$, and the remaining parameters define an interval that is disjoint. With respect to this interval, we partition Σ and U into two blocks,

$$\Sigma = \begin{bmatrix} \Sigma_1 & 0 \\ 0 & \Sigma_2 \end{bmatrix}, \quad U = \begin{bmatrix} U_1 \\ U_2 \end{bmatrix}, \tag{3.28}$$

such that the diagonals of $\Sigma_1 \in \mathbb{R}^{p\times p}$ are excluded from the interval and $\Sigma_2 \in \mathbb{R}^{(n-p)\times(n-p)}$ are contained in the intervals. The dimension of the reduced subspace will be $p+rs$, so we will fine-tune the interval and s to minimize this dimension while maintaining a target interpolation error bound.

In partitioned form, the form of the eigenvector of λ is

$$q(\lambda) = \begin{pmatrix} q_1 \\ (\lambda I - \Sigma_2)^{-1} U_2 w \end{pmatrix}, \tag{3.29}$$

where we no longer try to restrict the form of $q(\lambda)$ in the first block, expanding the subspace dimension to $p+r$. We approximate the Cauchy kernel with an interpolant of the form

$$q(\lambda) \approx \begin{pmatrix} q_1 \\ \sum_{i=1}^{s} c_i(\lambda)(\hat{\lambda}_i I - \Sigma_2)^{-1} U_2 w \end{pmatrix}. \tag{3.30}$$

We quantify the separation between the excluded poles and the interpolation domain by a parameter Δ defined as

$$\Delta = \frac{(\lambda_{\min} - \sigma_{\max})(\sigma_{\min} - \lambda_{\max})}{(\lambda_{\max} - \lambda_{\min})(\sigma_{\min} - \sigma_{\max}) + (\sigma_{\min} - \lambda_{\min})(\lambda_{\max} - \sigma_{\max})} \tag{3.31}$$

The details of this interpolation are not relevant to this application except for the interpolation points and error bounds. The interpolation points are determined by a Jacobi elliptic function, dn, and the quarter period, K, as

$$\Delta_1 = \lambda_{\max} - \lambda_{\min} + \sigma_{\min} - \sigma_{\max},$$

$$\Delta_2 = \sigma_{\min} - \lambda_{\max} + \Delta'(\lambda_{\min} - \sigma_{\max}),$$

$$\Delta' = \frac{\Delta}{1\sqrt{1-\Delta^2}},$$

$$m(z) = \frac{(\Delta_1 + \Delta_2)\lambda_{\max}(z+1) + (\Delta_2 - \Delta_2)\sigma_{\min}(z-1)}{2(\Delta_1 z + \Delta_2},$$

$$\lambda_i = m(dn(K(\sqrt{1-(\Delta')^2}\frac{2i-1}{2s}, \sqrt{1-(\Delta')^2})).$$

Two practical bounds on the maximum pointwise relative error are

$$\epsilon = \max||I - (\lambda - \Sigma_2)\sum_{i=1}^{s} c_i(\lambda)(\hat{\lambda}_i I - \Sigma_2)^{-1}||_2 \tag{3.32}$$

$$\epsilon \leq 2\exp\left(\frac{-\pi^2/2}{\ln(4/\Delta')}s\right) \leq 2\exp\left(\frac{-\pi^2/2}{\ln(8/\Delta)}s\right). \tag{3.33}$$

This is the small parameter that we should use to bound other errors more directly relevant to the eigenvalue problem.

Besides projection errors, we should also be wary of conditioning issues. If we now expand the eigenvectors for $\lambda \in [\lambda_{\min}, \lambda_{\max}]$ in an approximate space,

$$q(\lambda) \approx \begin{pmatrix} I & & ... & 0 \\ 0 & (\tilde{\lambda}_1 I - \Sigma_2)^{-1}U_2 & ... & (\tilde{\lambda}_s I - \Sigma_2)^{-1}U_2 \end{pmatrix} \begin{pmatrix} q_1 \\ \hat{w}_1 \\ \vdots \\ \hat{w}_s \end{pmatrix}, \tag{3.34}$$

we want to make sure that there isn't a large growth in the norm of the coefficient vector relative to $q(\lambda)$. We can independently orthonormalize the columns for each interpolation point by substituting $(\hat{\lambda}_i I - \Sigma_2)^{-1}U_2 \rightarrow (\hat{\lambda}_i I - \Sigma_2)^{-1}U_2[U_2^T(\hat{\lambda}_i I - \Sigma_2)^{-2}U_2]^{-1/2}$.

At an interpolation point, when $c_i(\hat{\lambda}_j) = \delta_{ij}$, the 2-norm of q and the interpolation vector will be equal. From the approximate factorization of the coefficient vector inherent in the interpolation, $\hat{w}_i \approx c_i(\lambda)w$, we can relate any remaining conditioning problems to large positive negative values in $c_i(\lambda)$ that cancel in forming the interpolation. Past studies show that this type of behavior is highly benign. Thus, while the basis is not orthonormal, we do not expect it to contribute significantly to degrading the numerical accuracy of the projection.

The final step in setting the projected problem is to fully calculate and simplify the matrix elements. To simplify the presentation and clarify the necessary computational work, we define some useful sub-matrices,

$$B_i = U_2^T(\hat{\lambda}_i I - \Sigma_2)^{-1} U_2,$$

$$C_i = U_2^T(\hat{\lambda}_i I - \Sigma_2)^{-2} U_2,$$

$$E_i = C_i^{-1/2} B_i,$$

$$F_{ij} = E_i C_j^{-1/2}.$$

The main trick used to simplify expressions are partial fraction expansions, which allow us to reduce all terms to E_i and F_{ij}. The left-hand matrix of the projected eigenvalue problem is

$$\begin{pmatrix} \Sigma_1 + U_1 D U_1^T & U_1 D E_q^T & \dots & U_1 D E_s^T \\ E_1 D U_1^T & \hat{\lambda}_1 I - F_{11} + E_1 D E_1^T & \dots & \frac{\hat{\lambda}_1 F_{1s} - \hat{\lambda}_s F_{s1}^T}{\hat{\lambda}_s - \hat{\lambda}_1} + E_1 D E_s^T \\ \vdots & \vdots & \ddots & \vdots \\ E_s D U_1^T & \frac{la\hat{m}bda_2 F_{s1} - \hat{\lambda}_1 F_{1s}^T}{\hat{\lambda}_1 - \hat{\lambda}_s} + E_s D E_1^T & \dots & \hat{\lambda}_s I - F_{ss} + E_s D E_s^T \end{pmatrix} \tag{3.35}$$

The right-hand matrix is

$$\begin{pmatrix} I & 0 & \dots & 0 \\ 0 & I & \dots & \frac{F_{1s} - F_{s1}^T}{\hat{\lambda}_s - \hat{\lambda}_1} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \frac{F_{s1} - F_{1s}^T}{\hat{\lambda}_1 - \hat{\lambda}_s} & \dots & I \end{pmatrix}. \tag{3.36}$$

If we are constructing B_i and C_i matrices for multiple partitions at once, we can consolidate and accelerate the arithmetic using fast Cauchy kernel summation techniques such as the fast multipole method. While we are making an *a priori* assumption that the projection basis is well-conditioned, we can test this a *posteriori* by examining the eigenvalues of the right-hand matrix.

Since all the projection work must be performed simultaneously to take advantage of fast methods, we must decide how to partition the problem efficiently before any detailed calculations of the eigendecomposition take place. Thus we need a sensible cost model to minimize over the choice of partitions. We choose a simple, greedy, divide-and-conquer strategy of repeatedly bisecting the approximation domain of eigenvalues and only consider the cost of the projected eigenvalue problems. Thus, the original unprojected problem is valid over $\mathbb{R}$ and size n with characteristic cost of n^3. For a target projective error ϵ, we can fine-tune $\lambda_{textcut}$ that defines a partition into two subintervals, $(-\infty, \lambda_{\text{cut}}]$ and $[\lambda_{\text{cut}}, \infty)$, and adjust the values of $\sigma_{\min}$ and $\sigma_{\max}$ for each subinterval to minimize the dimensions of the two projected problems, $p_1 + rs_1$ and $p_2 + rs_2$. We allow the bisection of the interval when we believe it will reduce the cost of solving the projected problem by satisfying

$$(p_1 + rs_1)^3 + (p_2 + rs_2)^3 < n^3. \tag{3.37}$$

We repeat this assessment the requested bisections until our cost model predicts that further bisection will not reduce the overall cost of the eigenvalue problem. At this point, we form all of the projected problems and solve for the various eigenvalue intervals simultaneously. We generally expect $O(n/(r \log \epsilon^{-1}))$ partitions of dimension $O(r \log \epsilon^{-1})$, leading to an $O(r^2 n \log^2 \epsilon^{-1})$ complexity for the eigensolver step. Pathological clustering on the diagonals of Σ to enforce a relative gap between specific diagonal entries or small clusters that can can be numerically resolved by the precision of the arithmetic. For double-precision arithmetic, an extremely small relative perturbation should suffice.

Assembling the global set of eigenvalues from the eigenvalues of the projected problems is straightforward as long as an eigenvalue on a boundary should be approximated well by both intervals sharing the boundary point, so you should simply be able to pick one and ignore the other without problems. If multiple degenerate eigenvalues appear at a boundary, then the degeneracy may be broken differently by each partition. It is important to extract the eigenvectors of this eigenvalue cluster from a single interval to avoid orthogonality problems. A conservative approach to dealing with eigenvalues at boundary points is to repartition and resolve to avoid the issues altogether.

3.2.2 Forward eigenvalue problem

Here we find the vector λ and $n \times r$ matrix Y that drive the residual of the eigenvalue problem down to a satisfactory error. This is done through a BFGS quasi-Newton optimization scheme. The keys to the forward problem are utilizing the structure of the eigenvectors, clever data storage and order of operations to minimize cost, and a heavy reliance on the FMM. Intuitively, it is the simpler of the two problems we solve for this scheme.

3.2.3 Objective function

The objective function for the forward eigenvalue problem is to minimize the following:

$$f(x) = \|\mathrm{diag}(X^T \tilde{A} X) - d\|_2 \tag{3.38}$$

Accurately, stably, and efficiently evaluating the objective function is a fundamental issue to the algorithm. We need to be able to do so in $O(r^2 n)$ time complexity consistently, while maintaining numerical accuracy and stability. This is done by leveraging the FMM and doing things in a specific order to bring down the cost. Noting the special structure of our eigenvector matrix from (2.1) this becomes:

$$f(x) = \|\mathrm{diag}((VY) \cdot (1/(d_i - \lambda_j))^T_{n \times n}(A + VV^T)(VY) \cdot (1/(d_i - \lambda_j))_{n \times n}) - d\|_2 \tag{3.39}$$

We can further simplify the equation to phrase it in terms of simpler subroutines.

$$f(x) = \|\mathrm{diag}(X^T D X + X^T V V^T x) \tag{3.40}$$

where we are aided by the structure in (2.3) but do not include it in equation (2.4) for compactness. This essentially comes down to two simpler sub-problems:

- computing column norms of our structured matrix
- computing matrix-vector multiplications with our structured matrix

Consider first the norm computation. We note that our structured matrix can be written as a generalized Cauchy-like matrix $C = (UV) \cdot (1/(x_i - y_j)$. It is possible to compute these in $O(r^2 n)$, but one must be very careful in the manipulation to ensure complexity is reduced.

An individual entry $C(i,j)$ can be represented as follows:

$$C(i,j) = \frac{U(i,:)V(:,j)}{x_i - y_j}$$

so the norm (squared) for the jth row would be

$$\|C(j,:)\|_2^2 = \sum_{j=1}^{n} \frac{(U(i,:)V(:,j))^2}{(x_i - y_j)^2} = \sum_{j=1}^{n} \frac{U(i,:)V(:,j)V(:,j)^T U(i,:)^T}{(x_i - y_j)^2}$$

Now if we let $A_j = V(:,j)V(,j)^T$, noting that these can all be precomputed in $O(r^2 n)$, then we have

$$\|C(j,:)\|_2^2 = \sum_{j=1}^{n} \frac{A_j U(i,:)U(:,i)^T}{(x_i - y_j)^2}$$

The difficulty here is to form $B_i = \sum_{j=1} \frac{A_j}{(x_i - y_j)^2}$ efficiently. To do this, let's consider $B_i \in \mathbb{R}^{r \times r}$ and its (k,k) element.

$$\begin{pmatrix} B_1(k,k) \\ \vdots \\ B_n(k,k) \end{pmatrix} = \begin{pmatrix} (x_1 - y_1)^{-2} & ... & (x_1 - y_n)^{-2} \\ \vdots & & \vdots \\ (x_n - y_1)^{-2} & ... & (x_n - y_n)^{-2} \end{pmatrix} \begin{pmatrix} A_1(k,k) \\ \vdots \\ A_n(k,k) \end{pmatrix}$$

This only costs $O(n)$ to do by the FMM. So if we let k run through $1 \leq k \leq r$, we see that it takes only $O(r^2 n)$ to compute all $B_i(k)$. Thus, we these ideas in mind, we can use the following algorithm to compute the entire eigenvector matrix-vector product in $O(r^2 n)$.

Algorithm 3 Evaluation of the forward problem objective function $f(z)$ with a given set of parameters d, V, Y, Λ

```
1: procedure fobj($d, V, Y, \Lambda$)
       Output: $f(z)$, where $z$ is represented by $d, V, Y, \Lambda$
2:     Initialize $f(z) = 0$
3:     Initialize vector $\mathbf{A} = \mathbf{0} \in \mathbb{R}^n$
4:     for k = $1, ..., n$ do
5:         $A(k) = V(:,k)V(:,k)^T$
6:     end for
7:     Initialize cell array $\mathbf{B} = \mathbf{0} \in \mathbb{R}^{r \times r \times n}$
8:     for  i = $1, ..., r$ do
9:         for  j = $1, ..., r$ do
10:            $B(i,j,:) = \mathrm{FMM}(K, A)$,      where $K(x,y) = (d(x) - \Lambda(y,y))^{-2}$
11:            for  k = $1, .., n$ do
12:                $f(z) = f(z) + (B(i,j,k)U(i,:)U(i,:)^T - \Lambda_i)$
13:            end for
14:        end for
15:    end for
16: end procedure
```

3.2.4 Derivative operators

Fundamental to the Newton scheme are the formation of the derivative operators for the optimization problems. As the optimization space being searched in is of dimension rn, keeping the complexity down is of fundamental importance. For the first derivative operator, this can be done by noting lots of zero components in the derivative and using the fast-multipole method throughout to approximate far-field interactions.

for the second derivative operator, we do not form the entire Hessian but form an HSS approximation of it. This is admissible because the off-diagonal blocks of this Hessian will be bounded by roughly $r \log n$ due to the displacement rank properties of Cauchy and Cauchy-like matrices, as his been previously proven in the literature. In this was we can do a fast randomized construction and moreover do HSS-eigendecompositions and linear system solves involving this Hessian matrix very quickly.

3.2.5 Inverse problem

The inverse objective function is:

$$g(x) = \|Q^T Q - I\|_2 \tag{3.41}$$

subject to the constraint $f(x) < \epsilon$

where $f(x)$ is the forward eigenvalue problem and ϵ is a pre-determined tolerance based on global tolerance.

We can solve this using the definition of the 2 norm as the largest eigenvalue (since the Hessian matrix will be symmetric positive definite. We thus use the power method and repetitively apply the matrix $Q^T Q - I$ to a random vector iteratively until it converges to an eigenvector. We then use the Rayleigh quotient to find the corresponding eigenvalue, which is our objective function. In practice, this randomly chosen vector is performing quite well for a variety of tests. However, it may be possible to choose a more specific initial vector based on the information in the problem to accelerate convergence. This has not been looked at rigorously yet and will be left for future work.

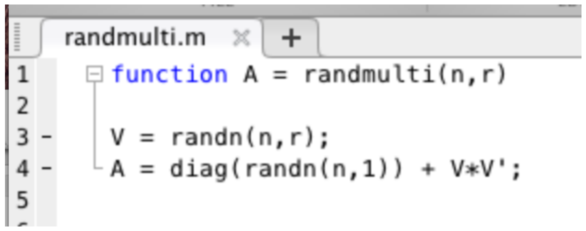
For the derivative operators, we do an approximation. We set the function as taking a finite power of products and this works in practice. In fact, in practice we have found that approximating this infinite product by only the square of the matrix is sufficiently good for finding the derivative vector and derivative matrix. In the future, we will do more rigorous analysis to see why this is the case. It is likely that for certain special cases, a higher number of products in the expression is required (likely when the eigenvalues are particularly clustered). In these cases, we may need to come up with alternative algorithms. But for now the heuristic of a quadratic approximation serves us well as an initial proof of concept for this inverse problem approach to stabilizing the eigenvectors.

This problem is solved much like the forward problem; with either a Newton or quasi-Newton approach. The numerical results shown in this thesis are with a quasi-Newton approach due to ease of implementation, but in practice one may get better performance for a large class (and more general class) of problems by using the full Newton method. The comparison of these two methods and their analysis will explored in depth in future work, but we present them both here to give the reader a full view of feasible possibilities in attacking this difficult problem. The algorithm shown below is for the full Newton method, as the quasi-Newton approach can be easily ascertained from it.

3.2.6 Algorithm

3.2.7 Numerical results

All tests run by testing matrix A generated by the following Matlab code:



```
function A = randmulti(n,r)

V = randn(n,r);
A = diag(randn(n,1)) + V*V';
```

Algorithm 4 Multi-rank update algorithm

```
1: procedure MR-update($d, V$)
       Output: $Y, \hat{V}, \Lambda$
2:     Solve for eigenvalues $\Lambda$ using spectrum slicing
3:     Solve for Sylvester vectors $Y$ using structured Newton's method
4:     Solve for minimum of unconstrained problem: $\beta_\mu(Y) = f(Y) + \mu\varphi(Y)$
5:     while $\|\nabla(\beta_\mu)\| >$ tol do
6:         Construct HSS approx. to Hessian $\tilde{H}_1$ (FMM, randomized sampling)
7:         Compute eigendecomposition $\tilde{H}_1 = Q\Lambda Q^T$
8:         Modify $\Lambda$ as necessary to preserve positive definiteness
9:         $p = Q\Lambda^{-1}Q^T(-\nabla(\beta_\mu))$
10:        Compute step length $\alpha$ with Wolfe Line Search
11:        $Y \longleftarrow Y + \alpha p$
12:    end while
13:    Solve for perturbed rank-r vectors $\hat{V}$ using structured Newton's method
14:    Solve for minimum of unconstrained problem: $\beta_\mu(\hat{V}) = g(\hat{V}) + \mu\varphi(\hat{V})$
15:    while $\|\nabla(\beta_\mu)\| >$ tol do
16:        Construct HSS approx. to Hessian $\tilde{H}_2$ (FMM, randomized sampling)
17:        Compute eigendecomposition $\tilde{H}_2 = Q\Lambda Q^T$
18:        Modify $\Lambda$ as necessary to preserve positive definiteness
19:        $p = Q\Lambda^{-1}Q^T(-\nabla(\beta_\mu))$
20:        Compute step length $\alpha$ with Wolfe Line Search
21:        $\hat{V} \longleftarrow \hat{V} + \alpha p$
22:    end while
23: end procedure
```

3.3 Sparse matrices

In this section we briefly note that by invoking a well-known result from graph theory to show that class of problems in which our family of algorithms can be

Table 3.8.
Complexity for performing a multi-rank update, residual tolerance set to 1e-6, orthogonality tolerance set to 1e-15

r \n	2000	4000	8000	16000
10	1.61e09	3.27e09	6.49e09	1.30e10
20	6.44e09	1.29e10	2.55e10	5.19e10
40	2.50e10	5.21e10	1.09e11	2.24e10
80	1.03e11	2.11e11	4.13e11	8.20e11

Table 3.9.
Accuracy for performing a multi-rank update, residual tolerance set to 1e-6, orthogonality tolerance set to 1e-15, r held constant at $r = 20$

n	2000	4000	8000	16000
γ	7.41e-09	4.33e-09	8.90e-09	1.02e-08
θ	8.64e-16	7.94e-16	8.13e-16	9.88e-16

effectively extended to the large class of sparse discretized problems. The work of Spielman and Tang [77] states that for any planar graph, one can constructively prove the existence of a separator of order $O(\sqrt{n})$. This allows us to make the following proposition.

Proposition 3.3.1 *Let A be a symmetric matrix resulting from the discretization of a 2-dimensional $n \times n$ graph. Then a separator can be found so that the HSS rank of A is at most $O(\sqrt{n})$.*

If the standard dense divide-and-conquer algorithm is applied to a two-dimensional sparse problem, the computational complexity would be roughly $O(n^2)$, following the complexity analysis in [88]. While this is better complexity than many existing algorithms, if the algorithm and complexity analysis are examined more carefully the cost can be decreased dramatically. For example, consider one of the computational bottlenecks of the algorithm, matrix-matrix multiplications of R generators in the divide portion of the algorithm in order to recursively update the B generators throughout the dividing process. From [88], we have that the cost is:

$$\xi_1 = \sum_{l=1}^{l_{\max}} 2^l \sum_{\tilde{l}=l+1}^{l_{\max}} 2^{\tilde{l}-l} \cdot (5 \cdot 2r^3) \approx 16r^3 \cdot 2^{l_{\max}} l_{\max} = O(r^2 n \log n).$$

However, this assumes that each R generator is treated as an unstructured dense matrix. In fact, in the sparse case, the R generators are just permutation matrices, and thus they can be multiplied at much smaller cost. It can similarly be shown for other bottleneck stages of the divide process that the computational complexity is reduced asymptotically by a factor of r for the case of sparse matrices. The complexity for conquer is unchanged asymptotically, though in practice this stage is faster for sparse matrices due to deflation. This implies that our algorithm can compute an eigendecomposition of any sparse 2-dimensional discretized problem in $O(n^{3/2} \log^2 n)$ complexity, which is a major improvement over many existing algorithms for this class of problem. This will justified numerically in Section 5.

As a further justification of the decreased computation of R generator computations, consider the HSS generator structure of a general banded matrix. As it has been show by several algorithms such as [17] that a sparse matrix can be transformed to a banded form in linear time, we do not lose any generality. Suppose that A is banded with blocks A_{jj} on the diagonal and $A_{j,j+1}$ on the first block superdiagonal, then the HSS generators look like [91]

$$D_{\mathbf{i}} = \begin{pmatrix} A_{j-1,j-1} & A_{j-1,j} & \\ A_{j,j-1} & A_{j,j} & A_{j,j+1} \\ & A_{j+1,j} & A_{j+1,j+1} \end{pmatrix}, \quad U_i = \begin{pmatrix} I & 0 \\ 0 & 0 \\ 0 & I \end{pmatrix},$$
$$R_{\mathbf{c}_1} = \begin{pmatrix} I & 0 \\ 0 & 0 \end{pmatrix}, \; R_{\mathbf{c}_2} = \begin{pmatrix} 0 & 0 \\ 0 & I \end{pmatrix}, \quad B_{\mathbf{c}_1} = \begin{pmatrix} 0 & 0 \\ A_{j+1,j+2} & 0 \end{pmatrix},$$

where the zero and identity blocks have sizes bounded by the half bandwidth.

We also note that for many 3-dimensional discretized problems, a separator of order $O(n^{2/3})$ can be found. Though this is not true in general. Nonetheless, this means that for large classes of 3-dimensional discretized problems, our algorithm can be used to compute a structured eigendecomposition in $O(n^{5/3}\log^2 n)$ computational cost, which is again faster than many current common approaches.

3.4 Extensions

In this section we extend to the superfast divide-and-conquer algorithm to non-symmetric and generalized eigenvalue problems. In future, we hope to do even further extensions.

3.4.1 Superfast singular value decomposition

For any matrix A of size $n \times m$ with $n \geq m$, there exists a singular value decomposition (SVD),

$$A = U\Sigma V^T \tag{3.42}$$

where U is an $m \times n$ orthogonal matrix, V is an $n \times n$ orthogonal matrix, and $\Sigma = \mathrm{diag}(\sigma_1, ..., \sigma_n)$ with $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_n \geq 0$.

As a straightforward extension to the algorithm in [88], we can quickly compute the SVD of any rank-structured A by doing an analogous divide procedure to that discussed in Section 2 and [88], solving for the SVD of a block-diaogonal matrix, and then recursively doing rank-one updates to the SVD. The recursion is very similar to that found in Section 2 of [88] so we only briefly discuss how it differs from the symmetric eigenvalue problem case, and we also clarify how to quickly and stably compute the SVD of the modified block-diagonal matrix and the rank-r updates.

Let i be a non leaf node of $\mathcal{T}$, and DC is applied to D_1. If $i = k$, then we divide the entire matrix A.

$$D_i = \begin{pmatrix} D_{c_1} & \\ & D_{c_2} \end{pmatrix} + \begin{pmatrix} U_{c_1} & \\ & U_{c_2} \end{pmatrix} \begin{pmatrix} & B_{c_1} \\ B_{c_2} & \end{pmatrix} \begin{pmatrix} V_{c_1}^T & \\ & V_{c_2}^T \end{pmatrix}$$

We avoid compiuting an eigendecomposition of $\begin{pmatrix} & B_{c_1} \\ B_{c_2} & \end{pmatrix}$, which would result in an unnecessarily computationally expensive rank-2r update to diag(D_{c_1}, D_{c_2}). Instead we write our low rank update as follows:

$$D_i = diag(\tilde{D}_{c_1}, \tilde{D}_{c_2}) + Y_i Z_i^T, \quad \text{where} \tag{3.43}$$

$$\tilde{D}_{c_1} = D_{c_1} - U_{c_1} B_{c_1} V_{c_1}^T, \quad \tilde{D}_{c_2} = D_{c_2} - U_{c_2} B_{c_2} V_{c_2}^T, \tag{3.44}$$

$$Y_i = \begin{pmatrix} U_{c_1} B_{c_1} \\ U_{c_2} B_{c_2} \end{pmatrix}, \quad Z_i = \begin{pmatrix} V_{c_1} \\ V_{c_2} \end{pmatrix}. \tag{3.45}$$

We can then find the SVD of the original matrix A by solving for the SVD of the block-diagonal matrix formed from updating the D generators and then recursively doing rank-one updates to the SVD, until we have the SVD of the original matrix A.

The first step of the conquer process is to solve for the SVDs of the leaf-level updated D_i blocks. Similarly to the eigendecomposition case, given the small size of these blocks, a relatively expensive (cubic complexity in time) method can be chosen and the global cost will still be near-linear. For this reason, we choose a method which is cubic in time but very accurate, stable, and requiring only very simple operations;

the algorithm of Lawson, Hanson, and Chan [58]. The singular vectors at the leaf level will be completely unstructured unlike in the latter stages of the algorithm. But this does not affect the global efficiency of the algorithm.

The key step in the conquer section of the algorithm is rank-r update to the SVD. In this paper, this is done by r rank-one updates to the SVD, as this allows us to utilizes lots of existing formalisms. In particular, we rely on the SVD update algorithm presented in [12]. Here we summarize the main ideas ideas found in [12] and then discuss how they can be utilized in our general SVD algorithm.

Suppose that $A = U\Sigma V^T$ is a singular decomposition and vw^T a rank-one update. We know that the singular values and left vectors $\hat{A} = A + vw^T$ can be solved by solving the eigenproblem for $\hat{A}\hat{A}^T$ and the right singular vectors can be found by $\hat{A}^T\hat{A}$. We aim to write this in the form

$$\hat{A} = A + v_1v_1^T + v_2v_2^T,$$

so that we can apply the well-developed theories of the rank-one update to the symmetric eigenvalue problem. We then note that

$$\hat{A}\hat{A}^T = (U\Sigma V^T + vw^T)(V\Sigma U^T + wv^T),$$

$$\hat{A}\hat{A}^T = AA^T + U\Sigma V^T wv^T + vw^T V\Sigma U^T (w^T w)vv^T.$$

Similarly, we we have that

$$\hat{A}^T\hat{A} = (V\Sigma U^T + wv^T)(U\Sigma V^T + vw^T),$$

$$\hat{A}^T\hat{A} = A^T A + V\Sigma U^T vw^T + wv^T U\Sigma V^T + (v^T v)ww^T.$$

Using strategies from [12, 41] we can take Schur decompositions and write

$$\hat{A}\hat{A}^T = AA^T + vQ\rho_1 Q_1^T v^T + \hat{w}Q_1\rho_2 Q_1^T \hat{w}^T = AA^T + v_1v_1^T + v_2v_2^T,$$

where $v_1 = \sqrt{-\rho_1}vQ_1$ and $v_2 = \sqrt{\rho_2}\hat{w}Q_1$. Similarly we have that

$$\hat{A}^T\hat{A} = A^T A + wQ_2\sigma_1 Q_2^T w^T + \hat{v}Q_w\sigma_2 Q_2^T \hat{v}^T = AA^T + \rho_1 v_1v_1^T + \rho_2 v_2v_2^T,$$

where $v_3 = \sqrt{-\sigma_1} w Q_2$ and $v_3 = \sqrt{\sigma_2} \hat{v} Q_2$.

This allows us to use the strategies from the conquer section of [88] to design a very efficient and stable algorithm of a superfast-DC-SVD in a straightforward way. The asymptotic cost is the same as in the symmetric eigenvalue problem case, the complexity is again $O(r^2 n \log 2) + O(rn \log^2 n)$ with slightly under double the total cost of the eigendecomposition algorithm.

3.4.2 Superfast generalized eigenvalue problems

We know that historically the tridiagonal divide-and-conquer algorithm has the ability to be extended to the generalized eigenvalue problem $Ax = \lambda \mathcal{B} x$, in which both A and $\mathcal{B}$ are symmetric tridiagonal matrices and moreover $\mathcal{B}$ is a positive definite matrix [28, 85]. The algorithms in [28, 85] provide two well known strategies to generalized eigenvalue problems for structured matrices. We will first summarize these two methods, then give an explanation as to why the rank structured case does not admit these generalizations, and finally provide an alternative approach.

The main concept in [28] is to observe that because the matrix is tridiagonal, our update problem $Q^T(A - \lambda\mathcal{B})Q = (D - yy^T) - \lambda(I - ww^T)$ is able to be expressed as $Q^T(A - \lambda\mathcal{B})Q = (D - \epsilon ww^T) - \lambda(I - ww^T)$, where ϵ is a constant, i.e.,

$$\mathcal{B} = \begin{pmatrix} A_1 & 0 \\ 0 & A_2 \end{pmatrix} - \epsilon v v^T. \tag{3.46}$$

Which gives us an extremely similar approach to the standard tridiagonal divide-and-conquer approach. Suppose that subproblems $A_i x = \lambda \mathcal{B}_i x, \quad i = 1, 2$ have been computed, then we obtain the following eigendecompositions:

$$Q_1^T(A_1 - \lambda\mathcal{B}_1)Q_1 = \Lambda_1 - \lambda I_s,$$

$$Q_2^T(A_2 - \lambda\mathcal{B}_2)Q_2 = \Lambda_2 - \lambda I_{n-2}.$$

We then have $Q = Q_1 \oplus Q_2, \quad \Lambda = \Lambda_1 \oplus \Lambda_2$ with $w = Q^T v$ which gives us that:

$$Q^T(A - \lambda\mathcal{B})Q = (\Lambda - \epsilon ww^T) - \lambda(I - ww^T)$$

This yields a secular equation, however there are key differences to the ones in [64,88]. Suppose that $f(\lambda)$ is the standard secular equation and then further suppose that $g(\lambda)$ is the equation which solves the secular equation subroutine of the generalized eigenvalue problem that we wish to solve.

$$g(\lambda) = (\epsilon - \lambda)^{-1} f(\lambda) = \sum_{i=1}^{n} \frac{w_i^2}{d_i - \lambda} - \frac{1}{\epsilon - \lambda}.$$

In the general rank-structured case a similar method is possible yet it is not computationally feasibly in less than $O(n^2)$ complexity. We can see that this by examining how the matrix $\Lambda - \epsilon I$ will be replaced by $\Lambda - yy^T$. However because this is no longer a diagonal matrix, it will increase the evaluation cost for the secular equation from linear to quadratic complexity.

The other well known approach to the problem is to apply a transformation to both sides of the matrix equation which yields a standard eigenvalue problem which can be solved by canonical techniques. Since $\mathcal{B}$ is an SPD matrix, we are able to factorize

$$Ax = \lambda LL^T x. \tag{3.47}$$

Then by first pre-multiplying by the inverse of the Cholesky factors then post-multiplying, an equivalent problem arises,

$$\tilde{A}x = L^{-1}AL^{-T}x = \lambda x. \tag{3.48}$$

The work most notable for the tridiagonal case is by [85]. There they prove that if A and $\mathcal{B}$ are tridiagonal matrices, we then have that the matrices L^{-1} and L^{-T} in the above equation are actually quasiseparable. The product $\tilde{A}$ is thus also quasiseparable. In theory we could extend this concept to HSS matrices. Notice that according to [**?**], there is a feasible way to accurately and efficiently compute part of an inverse of an HSS matrix. However, since the framework described above would require the entirety of the inverse matrix, yet this would be a stability nightmare to actually compute. Thus neither of the two well known approaches described here are practical to generalize to the HSS case.

As we know how important both the efficiency and stability features are to our algorithmic design, we decided to propose a third more robust method, which is based on the method of simultaneous diagonalization purposed by [62]. Which yields a stable approach that is consistently found to be still near-linear in computational complexity. We Follow the general framework in [62], which considers the simple case where a matrix is low rank. However, our work considers the more general case where the matrix need only have a low rank-property. But the same framework can be employed. It should be noted that [62] uses a canonical eigensolver for the diagonalization, so employing a superfast eigendecomposition algorithm causes a significant speedup.

In the first major computation of the algorithm one computes a standard superfast eigendecomposition for the matrix $\mathcal{B}$ noting that $\mathcal{B} = X^T \Sigma X$, where X is used to denote the eigenvectors in structured form and Σ is the eigenvalues matrix which due to symmetry properties will all be real and positive as $\mathcal{B}$ is an SPD matrix. Due to this property, one can easily form the inverse square root of the diagonal matrix Σ and then compute $X' = X\Sigma^{-1/2}$. In the next step we compute an intermediate matrix C by transforming A by X through the use of multiplicative pre and post factors. So this gives us that $C = (X')^T AX'$. As we are multiplying HSS matrices with one another, the process can be done very fast without rank structured being altered. Below we give a sketch of a lower bound proof for the HSS rank of the matrix C. We also discuss how this affects the overall computational complexity of our algorithm. After these steps, we must simply compute a second superfast eigendecomposition, this time finding eignevalues and eigenvectors of C. We obtain $C = (X'')^T \Sigma' X''$. Once we have finished this computation, one easily computes the eigenvector and eigenvalue solution of the generalized eigenvalue problem we originally wanted to solve as $Q = X'X''$ and respectively $\Lambda = Q^T AQ$. The major framework for our superfast generalized eigendecomposition algorithm can be found in the table in following section. The cost can be shown both analytically and computationally to be similar to that of the standard eigenvalue algorithm and the singular value decomposition algorithm. As will be shown in Section 4 and Section 5, this algorithm can be quite effective in quickly solving certain classes of discretized problems.

3.4.3 Switching levels

While the algorithm in [88] is very fast, scaling nearly-linearly in time and storage, significant computational savings can be made if three *switching levels* are employed. A *switching level* is where for levels $l_{\max}$ to l_α of the tree using dense linear algebra is more cost-effective than structured linear algebra due to the small block size. This is important for efficient implementation for the superfast eigendecomposition, but it is even more crucially important for the generalized eigendecomposition and sparse matrix examples given in Section 3, and the discussion of use of switching levels is also revisited in that section.

As an example, consider the direct eigendecomposition computations at the leaf level of a matrix. Suppose the matrix size is n and each leaf level block has a small constant size equal to m. The cost to compute a full eigendecomposition of each block will be

$$\sum_{i=1}^{(n/m)} 5m^3 + O(1) = O(n).$$

where the asymptotic constant is quite small. If on the other hand, we were to employ structured linear algebra, we would still get $O(n)$ cost, but with a much larger constant. It is thus natural to solve for the following three constants each time when doing a superfast eigendecomposition:

- m, the leaf level block size. In HSS linear solvers [92], the optimal value for this parameter can be shown to be $2r$, where r is the HSS rank of the matrix. However, for HSS eigensolvers, a slightly larger value is optimal.

- l_α is where we start to structure the intermediate eigenvector matrices. Below this level, they are stored densely, above we use Cauchy-like and Householder representations. The benefit of using dense linear algebra is not having a large constant associated with FMM but a larger value of l_α will affect all levels as the application of eigenvector matrices is recursive.

- l_β, is where fast root-finding techniques are employed. At lower levels of the HSS tree, it is more effective to use an $O(n^2)$ bisection scheme to solve for the updated eigenvalues than inverse interpolation-based root-finding. In practice setting $l_\beta \approx l_\alpha$ gives good performance but is not quite optimal.

The idea of bisection and inertia evaluation as a means of eigenvalue solutions can be found in many places such as [72]. More recently, an algorithm in [90] combined these ideas with HSS structures to design an $O(n^2)$ algorithm for computing all eigenvalues of a symmetric matrix. However, given that our subproblems are effectively diagonal plus rank-r, a much simpler algorithm can be used and is preferable to optimize one's implementation.

These three parameters are highly coupled, and the best way to solve for them is to numerically solve a simple optimization problem for total floating point operations where r is set to be constant and m, l_α, and l_β make up the solution vector. To illustrate this, we examine the simple case when only one switching level parameter, say m, is to be solved for. For an optimal switching level we want the computational costs below and above the switching level to be equal. The actual expression is longer and can be derived easily from the detailed complexity analysis in Section 3.1 of [88], but to simplify the problem we can approximate this equivalence as

$$\alpha m^2 n = \beta r^2 n \log_2 m, \tag{3.49}$$

where α is the constant associated with the cubic-time dense eigendecompositon algorithm used at the leaf-level and β is roughly the constant associated with the whole conquer section of the algorithm. Taking a derivative of this and setting to zero gives no closed form solution for the switching level. When all complexity terms are included and all three coupled switching levels are included, finding analytic expressions that express or even well-approximate optimal switching levels becomes even more difficult, and a simple numerical optimization solution becomes a more viable and attractive way of solving this problem.

Fortunately, this optimization problem for all three parameters can be set up easily using the complexity analysis in Section 3.1 of [88] and this problem can be solved numerically as a simple pre-computation and takes constant time to solve. Conversely, it can be shown to have great computational benefit. While the benefit is problem dependent on the HSS rank of the matrix and moreover the distribution of rank of the matrix, a significant speedup is usually noticed.

4. ANALYSIS FOR THE STRUCTURED EIGENVALUE PROBLEM

While many readers of this thesis may focus on the impressive scaling results in the previous section or the promising applications in the section that follow, we encourage a thorough read of this analysis discussion which we find to be a key part to this research program. While much work has been done in the past decade studying how the use of hierarchical matrices (and off-diagonal compression in general) affects the solution of linear systems, there is little work in this area for the analogous matrix eigenvalue problem. We seek to lay a foundational framework for how to approach such a problem theoretically, which we hope to follow up on with a significant amount of future work. We start with some basic convexity results which guide some algorithmic choices used for the multi-rank update sub-routines. We then review some detailed complexity arguments that show the near-linear scaling that are hallmarks of our class of algorithms. Finally we take a deep dive into the algorithms' accuracy; both at a global level and looking at the stability and accuracy of individual components such as a 2×2 block solution and in solving the secular equation.

4.1 Convexity analysis

A fundamental issue to the analysis of the multi-rank update problem algorithms is that of convexity. In order to show that the algorithm will converge to a suitable global maximum, we must show that that the optimization problems are in fact convex. This require two things; that the second derivative of the Hessian matrix for these problems is positive semi-definite, and that the feasible regions are convex. We prove these properties here.

4.1.1 Higher derivatives in forward eigenvalue problems

In this case the feasible region for optimization is unconstrained, so it is trivial that the region is convex. It suffices to show that the higher derivative operators have non-negative eigenvalues. We can prove this directly from the definition of convexity.

Proof.

$$f(x) = \|\text{diag}(X^T DX + X^T VV^T x)\|_2 \tag{4.1}$$

Thus $f(\lambda x + (1-\lambda)y) \leq f(\lambda x) + f((1-\lambda)y)$

(by the triangle inequality of vector norms)

Similarly $f(\lambda x) + f((1-\lambda)y) = \lambda f(x) + (1-\lambda)f(y)$

(by the homogeneity property of vector norms)

As this holds for any $x, y \in \mathbb{R}^n$, we have that f is convex $\blacksquare$

4.1.2 Higher derivatives in inverse eigenvalue problems

In the case of the inverse problem, we have a constrained optimization region, but the convexity of this region is again trivial because it is a norm bound. Thus all we must do again is show that the higher derivative operators have non-negative eigenvalues. Again, we can prove this directly from the definition of convexity.

Proof.

$$g(x) = \|Q^T Q - I\|_2 \tag{4.2}$$

Thus $g(\lambda x + (1-\lambda)y) \leq g(\lambda x) + g((1-\lambda)y)$

(by the triangle inequality of vector norms)

Similarly $g(\lambda x) + g((1-\lambda)y) = \lambda g(x) + (1-\lambda)g(y)$

(by the homogeneity property of vector norms)

As this holds for any $x, y \in \mathbb{R}^n$, we have that g is convex $\blacksquare$

4.2 Complexity analysis

A fundamental feature of all of the algorithms in this thesis is that they scale near-linearly in run time and storage complexity. While the numerical results presented herein give a good indication of this, it is also important that we present theoretical proofs of these properties. In this section, we show the theoretical asymptotic computational complexity for the two main algorithms in this thesis; the multi-rank update algorithm and the superfast divide-and-conquer algorithm.

4.2.1 Mulit-rank update

The storage for the multi-rank update is the most essential part of the complexity, and thus we go over it in detail. The first object that needs to be stored is the structured eigenvector matrix. This consists of:

- Y, the Sylvester matrix, which contains up to rn nonzero elements
- λ, the eigenvalue matrix, which contains up to n nonzero elements
- V, the rank-r update matrix, which contains up to rn nonzero elements
- d, the initial eigenvalue vector, which contains up to n nonzero elements
- a small normalization matrix

All of this combined gives $O(rn)$ data for the eigenvector representation. This is also the cost of the first derivative vector for the objective functions. The most expensive cost is that of storing the Hessian matrix. Note that this is theoretically

$O(n^2)$ data, but we can take advantage of the rank structure and perform a randomized HSS construction on this matrix. As noted in the literature, the cost of this construction is $O(r^2 n \log n)$, where k is the largest off-diagonal rank. Thus in this case the Hessian formation has a cost of $O(r^2 n \log n)$, which becomes the dominant global cost for the algorithm.

4.2.2 Superfast divide-and-conquer

To facilitate the understanding the algorithm complexity analysis, the framework of the algorithm is given below. Here, it is assumed that the HSS tree $\mathcal{T}$ is a complete binary tree with $l_{\max}+1$ levels, with the root at level 0 and the leaves at level $l_{\max}$.

Table 4.1.
Major operations in the superfast DC algorithm and their complexity

Outer-most loop	Inner-most loop	Operation	Complexity subtotal
$l = 1 : l_{\max}-1$	Descendents $\mathbf{j}$ of $\mathbf{i}$	Updating $B_{\mathbf{j}}$ generators	$\xi_1 = O(r^2 n \log n)$
	Leaf descendants $\mathbf{j}$ of $\mathbf{i}$	Updating $D_{\mathbf{j}}$ generators	$\xi_2 = O(r^2 n \log n)$
$l = l_{\max}$	Leaves $\mathbf{i}$	Eigendecomposition of $D_{\mathbf{i}}$	$\xi_3 = O(r^2 n)$
$l = l_{\max}-1 : 0$	ith rank-1 update $(i = 1, \dots, r)$	Intermediate eigenmatrix-vector product	$\xi_4 = O(rn \log^2 n)$
		Rootfinding	$\xi_5 = O(rn \log n)$
		Finding perturbed eigenproblem	$\xi_6 = O(rn \log n)$
		Normalization	$\xi_7 = O(rn \log n)$

The dividing stage involves three nested loops. The outer-most loop is a top-down sweep through the levels l of the HSS tree, the next loop is through the nodes $\mathbf{i}$ at a given level l, and the inner-most loop is through each descendent $\mathbf{j}$ of $\mathbf{i}$.

The conquering stage is also done by three nested loops. The outer-most loop is a bottom-up sweep through the levels l of the HSS tree, the next loop is through the nodes $\mathbf{i}$ at a given level, and the inner-most loop is through each of the r rank-one updates. At each step, we complete four tasks. The first task is to form the vector v in $\tilde{\Lambda} + vv^T$ as in other work. The next step is to approximately solve for the eigenvalues λ of $\tilde{\Lambda} + vv^T$ by finding the roots of the secular equation. The third step is to solve the perturbed eigenvalue problem to find a vector $\hat{v}$ such that λ is an exact eigenvalue of $\tilde{\Lambda} + \hat{v}\hat{v}^T$. Finally, find the orthogonal eigenmatrix of $\tilde{\Lambda} + vv^T$. This eigenmatrix has a Cauchy-like form defined by $\hat{v}$, s, $\mathrm{diag}(\tilde{\Lambda})$, and λ's.

Before presenting the complexity analysis, we mention the two types of off-diagonal ranks involved. One type is the off-diagonal rank bound r of A, which is problem dependent. Another type includes the off-diagonal ranks of each matrix implicitly involved in the FMM matrix-vector evaluation.

For the kernel functions $\phi(x)$ involved in this work, it can be shown that $\tilde{r}$ is bounded, so that each FMM matrix-vector product can be implemented in linear complexity.

We now derive analytically the complexity of our algorithm. The numerical results give a view of how the algorithm scales in practice. The results in this section and the numerical results agree asymptotically. The framework has a summary of the complexity of the major computations and introduces notation. As often done in HSS algorithms [92, 95], we assume that the leaf level D generators have size $2r$, all the R, B generators have size r, and the HSS tree has $l_{\max} + 1 \approx \log(\frac{n}{2r})$ levels.

During the dividing stage, at each level l of the HSS tree, there are 2^l nodes $\mathbf{i}$. For each $\mathbf{i}$ at level l, we update $B_{\mathbf{j}}$ generators associated with each descendant $\mathbf{j}$ of $\mathbf{i}$. There are $2^{\tilde{l}-l}$ nodes $\mathbf{j}$ at level $\tilde{l} = l+1, \ldots, l_{\max}$. Four more matrix multiplications and one matrix subtraction is needed for each $\mathbf{j}$. Thus, the total cost to update all the $B_{\mathbf{j}}$ generators is

$$\xi_1 = \sum_{l=1}^{l_{\max}} 2^l \sum_{\tilde{l}=l+1}^{l_{\max}} 2^{\tilde{l}-l} \cdot (4 \cdot 2r^3) \approx 16r^3 \cdot 2^{l_{\max}} l_{\max} = O(r^2 n \log n),$$

where the low order terms are dropped (this is done similarly later).

The update of the $D_{\mathbf{j}}$ generators at level $l_{\max}$ follows the update of all the $B_{\mathbf{j}}$ generators. The cost is

$$\xi_2 = \sum_{l=1}^{l_{\max}} 2^l \cdot 2^{l_{\max}-l}(2 \cdot 2r^3) = O(r^2 n \log n).$$

At the leaf level we compute the eigendecomposition of $D_{\mathbf{i}}$ for each leaf $\mathbf{i}$. The total cost is

$$\xi_3 = \frac{n}{2r}\left(2 \cdot (2r)^3\right) = O(r^2 n).$$

During the conquering stage, for each node $\mathbf{i}$ at each level l of the HSS tree, a sequence of operations are performed to find the eigenvectors.

One operation is to perform the intermediate eigenmatrix-vector multiplication in terms of $Q_{\mathbf{i}_1}, \ldots, Q_{\mathbf{i}}$ associated with $\mathbf{i}$ and its descendants. Each FMM application involved here has linear complexity $O(\frac{n}{2^{\tilde{l}}})$, where $\frac{n}{2^{\tilde{l}}}$ is the size of $Q_{\mathbf{j}}$ for $\mathbf{j}$ at level $\tilde{l} = l, l+1, \ldots, l_{\max}$. Here, $Q_{\mathbf{j}}$ is further given by r Cauchy-like matrices. This cost of the intermediate eigenmatrix-vector multiplication associated with $\mathbf{i}$ is thus

$$\sum_{\tilde{l}=l}^{l_{\max}} 2^{\tilde{l}-l} \cdot r \cdot O(\frac{n}{2^{\tilde{l}}}) = O(r\frac{n}{2^l}(l_{\max} - l)). \tag{4.3}$$

The subtotal for all $\mathbf{i}$ is

$$\xi_4 = \sum_{l=1}^{l_{\max}} 2^l \sum_{\tilde{l}=l}^{l_{\max}} 2^{\tilde{l}-l} \cdot r \cdot O(\frac{n}{2^{\tilde{l}}}) = O(rn \log^2 n).$$

Another operation is to solve r secular equations associated with each node $\mathbf{i}$. The cost with FMM for each secular equation is $O(\frac{n}{2^l})$. Thus, the subtotal is

$$\xi_5 = \sum_{l=0}^{l_{\max}-1} 2^l \cdot r \cdot O(\frac{n}{2^l}) = O(rn \log n).$$

The costs in the other operations are

$$\xi_6 = O(rn \log n), \quad \xi_7 = O(rn \log n).$$

Therefore, we obtain the total cost $\xi = \xi_1 + \cdots + \xi_7$ for the superfast DC algorithm. Clearly, if r is bounded, the cost ξ_4 for applying the intermediate eigenmatrices to vectors dominates the complexity. In general, the conquering stage costs a lot more than the dividing stage. In addition, by setting $l = 0$ (for $\mathbf{i} = \mathbf{k}$), we get the cost $\tilde{\xi}$ for applying the eigenmatrix Q of A to a vector. The storage for the $Q_{\mathbf{i}}$ matrices in terms of Cauchy-like forms can be easily counted. These results are summarized in the following theorem, where the result for a Toeplitz matrix is used.

Theorem 4.2.1 *Assume all the intermediate eigenvalues in the superfast DC scheme are uniformly distributed. The algorithm costs ξ to find all the eigenvalues and $\tilde{\xi}$ to apply the eigenmatrix to a vector, where*

$$\xi = O(r^2 n \log n) + O(rn \log^2 n), \quad \tilde{\xi} = O(rn \log n).$$

The storage for the eigenmatrix is

$$\sigma = O(rn \log n).$$

Specifically, if A is a banded symmetric matrix with finite bandwidth,

$$\xi = O(n \log^2 n), \quad \tilde{\xi} = O(n \log n), \quad \sigma = O(n \log n),$$

and if A is a symmetric Toeplitz matrix,

$$\xi = O(n \log^3 n), \quad \tilde{\xi} = O(n \log^2 n), \quad \sigma = O(n \log^2 n).$$

4.3 Accuracy analysis

In this section we first provide a uniform bound on eigenvalue accuracy of our algorithm that holds for any problem. We then show several very useful special cases that highlight that in most real-life we get much better accuracy than the uniform bound.

4.3.1 Uniform eigenvalue accuracy bound

Given that the eigenvalue algorithm in [88] often begins with a pre-processing step of approximating a symmetric matrix A with a symmetric HSS form $\tilde{A}$, it is important to rigorously understand the relationship between the compression of the off-diagonal blocks of A and the accuracy of the computed eigenvalues with respect to the the original matrix eigenvalue problem. While in general one can have a variable tolerance for different blocks of an HSS form $\tilde{A}$, for ease of discussion we assume here that each off-diagonal block has been compressed with a uniform tolerance τ.

In [88], a uniform bound is proved for such eigenvalue accuracies. While this bound is sharp, in many practical cases it is quite pessimistic. In this section we restate some of the major structured perturbation analysis results from [88] and then expand upon the discussion that paper to illustrate some common special cases where even stronger eigenvalue accuracy results hold. This has important practical implications, as it means for large classes of problems we can truncate the off-diagonal blocks very aggressively and still expect to recover very accurate eigenvalues.

Consider a block 2×2 form A that admits a one-level HSS approximation:

$$A \equiv \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \approx \tilde{A} \equiv \begin{pmatrix} D_1 & U_1 B_1 U_2^T \\ U_2 B_1^T U_1^T & D_2 \end{pmatrix}, \tag{4.4}$$

As shown in [88], we can derive a bound on how well the eigenvalues $\tilde{\lambda}_i$ of $\tilde{A}$ approximate those of A. Given an HSS construction where the off-diagonal block A_{12} is approximately equal to the compressed form $U_1 B_1 U_2^T$, we can write an explicit formulation for the error of matrix A, here denoted by E.

$$A_{12} = \begin{pmatrix} U_1 & \hat{U}_1 \end{pmatrix} \begin{pmatrix} B_1 & \\ & \hat{B}_1 \end{pmatrix} \begin{pmatrix} U_2^T \\ \hat{U}_2^T \end{pmatrix} = U_1 B_1 U_2^T + \hat{U}_1 \hat{B}_1 \hat{U}_2^T. \tag{4.5}$$

Thus,

$$A = \tilde{A} + E, \quad \text{with} \quad E = \begin{pmatrix} 0 & \hat{U}_1 \hat{B}_1 \hat{U}_2^T \\ \hat{U}_2 \hat{B}_1^T \hat{U}_1^T & 0 \end{pmatrix}.$$

This follows from the fact that the row bases U_i and $\tilde{U_i}$ are orthogonal, and thus that $\|E\|_2 = \|\hat{B}_i\|_2$. Thus it is easy to show the following lemma.

Theorem 4.3.1 *For A and $\tilde{A}$, suppose $||\hat{B}_1||_2 \leq \tau$. Then*

$$|\lambda_i - \tilde{\lambda}_i| \leq \tau.$$

In [88], the lemma is then used to derive a uniform bound on the eigenvalue accuracy relative to off-diagonal compression τ and the number levels l one uses in their HSS approximation. We state the theorem without proof here, as it is proven in [88].

Theorem 4.3.2 *Suppose a multilevel HSS approximation $\tilde{A}$ to A is constructed via truncated SVDs applied to the off-diagonal blocks of A, so that each B generator is obtained with the accuracy τ. Let l be the total number of levels (excluding the root) in the HSS tree. Then the approximation error matrix $E = A - \tilde{A}$ satisfies the following bound that is* attainable*:*

$$||E||_2 \leq l\tau. \tag{4.6}$$

Thus,

$$|\lambda_i - \tilde{\lambda}_i| \leq l\tau.$$

This bound is shown to be sharp in [88], but to be sharp the error matrix of an HSS approximation must have a very special, pathological, structure. In many practical situations, the uniform bound turns out to be quite pessimistic. Here we give a few important cases where stronger results can be given. In particular, two major strategies we employ to find tighter accuracy bounds are to examine the separation of the eigenvalues λ_i of A and to consider possible decay properties of the matrix A. Both strategies are briefly hinted at in [88], but we expand both discussions here.

4.3.2 Well-separated eigenvalues

For eigenvalues that are well separated, once can combine with results in [53] and show the following proposition.

Proposition 4.3.1 *Let $E = A - \tilde{A}$ be the HSS approximation error. Then for any eigenvalue λ_i of A satisfying $|\lambda_i - \lambda_{i+1}| > 2l\tau$, $|\lambda_{i-1} - \lambda_i| > 2l\tau$, we have*

$$|\lambda_i - \tilde{\lambda}_i| \leq ||Eq_i||_2, \tag{4.7}$$

where q_i is the eigenvector associated with λ_i.

As mentioned in [88], this result is particularly useful as it allows us to map the effect of off-diagonal truncation to eigenvalue accuracy through matrix vector products. By this we mean that the q_i in the above proposition can be found simply from the numerical eigenvector matrix. It is also important to note that in the case when eigenvalues are not well separated, we can still make improvements. If the clustering is tight enough, one may aggressively deflate the problem to a desired tolerance.

4.3.3 Decaying norm structure

A second important special case that is very common to structured matrix computations is when the off-diagonal blocks exhibit a decay property. For example the common matrix kernels $1/|x-y|$ and $1/|x-y|^2$ exhibit this property when the points are relatively uniformly distributed and arbitrary real values are chosen for the diagonal entries. As such, here we give some very useful error diminishing effects so that the approximation errors in some off-diagonal blocks have little impact on the accuracy of certain eigenvalues.

Following the eigenvalue perturbation analysis in [69], there is a useful *shielding effect* related to the compression of the off-diagonal blocks of A. That is, for certain eigenvalues λ of A originating from say A_{11}, the accuracy of λ is roughly shielded from the approximation error within the other subproblem A_{22} (λ is said to originate from A_{11} [69] in the sense that it is a certain continuous function of the perturbation). That is, the HSS approximation error δ in A_{22} appears in the error bound of λ like $O(\delta^2)$.

In particular, if the singular values of the off-diagonal blocks quickly decay to a desired accuracy τ, then a compact HSS form $\tilde{A}$ can be used to compute the eigenvalues with satisfactory accuracies. This type of problem is an important application of HSS methods. For example, when A results from the discretization of a kernel function that is smooth away from the diagonal singularity, then the off-diagonal blocks have a decay property, i.e., they quickly decay when they are farther away from the diagonal. If the off-diagonal blocks has the decay property, then there is also an attractive *multiplicative effect* [69] for the off-diagonal blocks. As an example, consider a two-level HSS approximation to a block 4×4 matrix $A = (A_{ij})_{4\times 4}$, where the decay property looks like

$$\begin{aligned} \|A_{i,i+1}\|_2 &\le \tau_1, & i &= 1,2,3, \\ \|A_{i,i+2}\|_2 &\le \tau_2 \ll \tau_1, & i &= 1,2, \\ \|A_{i,i+3}\|_2 &\le \tau_3 \ll \tau_2, & i &= 1. \end{aligned}$$

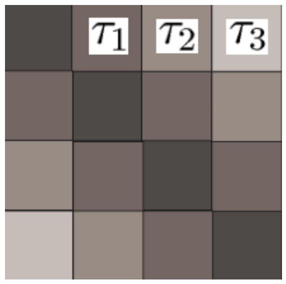


Fig. 4.1. an illustration of block 4×4 matrix with a decay property

For an eigenvalue λ of A originating from the leading block 2×2 principal submatrix $A_{1:2,1:2}$, let

$$\delta_1 = \min_{\hat{\lambda}\in\boldsymbol{\lambda}(A_{1:2,1:2})} |\lambda - \hat{\lambda}|, \quad \delta_2 = \min_{\hat{\lambda}\in\boldsymbol{\lambda}(A_{2:4,2:4})} |\lambda - \hat{\lambda}|, \quad \delta_3 = \min_{\hat{\lambda}\in\boldsymbol{\lambda}(A_{3:4,3:4})} |\lambda - \hat{\lambda}|,$$

Based on a derivation in [69],

$$\delta_2 \lessapprox \frac{1}{\delta_3}\tau_1(\tau_2 + \frac{1}{\delta_2}\tau_1^2).$$

When λ is separated from the eigenvalues in $\boldsymbol{\lambda}(A_{2:4,2:4})$ and $\boldsymbol{\lambda}(A_{3:4,3:4})$, the impact of τ_1 on the accuracy appears as $O(\tau_1^3)+O(\tau_1\tau_2)$ and is significantly diminished. Thus, τ_1 does not have to be very small. This also indicates that some extreme eigenvalues of A may be accurately approximated by those of the HSS form $\tilde{A}$.

However, in many cases the decay property is too weak to take advantage of the above results but large enough that performing full low-rank updates to the relevant symmetric eigenvalue sub-problem is not an efficient approach. In these cases, a refinement strategy can be employed successfully. Details of the refinement strategy and analysis are given below is Section 4.3.4.

4.3.4 Iterative refinement

We next introduce and prove the reliability of a major practical improvement over the algorithm in [88], the use of refinement. This applies to cases where the eigenvalues are not necessarily well-separated, and while they exhibit a decay property, the decay may be very weak. This is a very common situation in structured matrix computations.

For many problems, it is often the case that the singular values σ_i of A_{12} decay to around τ_1 but then decay very slowly. Now suppose τ_1 is close to the desired eigenvalue accuracy τ, but is slightly larger. Thus, if we compute an approximate eigenvalue $\tilde{\lambda}$ based on the rank-r update, and especially if $\tilde{\lambda}$ is close to its nearby eigenvalues, the accuracy in $\tilde{\lambda}$ may only be τ_1 and is not sufficient. A similar phenomenon occurs when the eigenvalues are *weakly clustered*, that is clustered enough where a full rank-one update would be inefficient and possibly unstable but traditional deflation techniques cannot be applied without sacrificing accuracy. Thus this refinement strategy is of essential importance to a practical implementation of our algorithm.

Our technique is to refine the accuracy by a reasonable amount, without the need to include any additional low-rank update beyond σ_r so as to save a significant amount of work. To illustrate the idea, consider a 2×2 problem

$$A = \tilde{A} + E \equiv \begin{pmatrix} 1 & \\ & 1+\tau_2 \end{pmatrix} + \tau_1 \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} \begin{pmatrix} u_1 & u_2 \end{pmatrix}, \tag{4.8}$$

where $\tau_1 \geq \tau_2 > \tau$ and $||\begin{pmatrix} u_1 & u_2 \end{pmatrix}^T||_2 = 1$ (if this norm is not 1, we can scale the vector and modify τ_1). Hence, the distance τ_2 between the two eigenvalues of $\tilde{A}$ as follows is not larger than $\|E\|_2$:

$$\tilde{\lambda}_1 = 1 + \tau_2, \quad \tilde{\lambda}_2 = 1. \tag{4.9}$$

Therefore, to reach the desired accuracy τ, the rank-one update matrix E is not supposed to be ignored. On the other hand, we can improve the accuracy of $\tilde{\lambda}_1$ and $\tilde{\lambda}_2$ based on the following refinement strategy without recomputing them via the rank-one update.

Consider a given tolerance $\tau < \tau_2 \leq \tau_1$. Let λ_1 and λ_2 be the eigenvalues of A, and

$$\hat{\lambda}_{1,2} = \begin{cases} 1 + \frac{1}{2}(\tau_1+\tau_2) \pm \frac{1}{4}\left(\tau_1+\tau_2+\sqrt{\tau_1^2+\tau_2^2}\right), & \text{if } |u_1| \leq |u_2|, \\ 1 + \frac{1}{2}(\tau_1+\tau_2) \pm \frac{1}{4}\left(\tau_1-\tau_2+\sqrt{\tau_1^2+\tau_2^2}\right), & \text{otherwise.} \end{cases} \tag{4.10}$$

Then for $i = 1, 2$,

$$|\lambda_i - \hat{\lambda}_i| \leq \begin{cases} \frac{1}{4}(\tau_1+\tau_2) - \frac{1}{4}\sqrt{\tau_1^2+\tau_2^2}, & \text{if } |u_1| \leq |u_2|, \\ -\frac{1}{4}(\tau_1-\tau_2) + \frac{1}{4}\sqrt{\tau_1^2+\tau_2^2}, & \text{otherwise.} \end{cases}$$

If furthermore, $\tau_1 = \tau_2$, then

$$|\lambda_i - \hat{\lambda}_i| < \begin{cases} \frac{3}{16}\tau_1, & \text{if } |u_1| \leq |u_2|, \\ \frac{3}{8}\tau_1, & \text{otherwise.} \end{cases} \tag{4.11}$$

Proof The eigenvalues of A are

$$\lambda_{1,2} = 1 + \frac{1}{2}(\tau_1+\tau_2) \pm \frac{1}{2}\sqrt{\tau_1^2 + 2\tau_1\tau_2(u_2^2 - u_1^2) + \tau_2^2}.$$

If $|u_1| \le |u_2|$, then

$$\lambda_1 \ge 1 + \frac{1}{2}(\tau_1+\tau_2) + \frac{1}{2}\sqrt{\tau_1^2+\tau_2^2},$$
$$\lambda_1 \le 1 + \frac{1}{2}(\tau_1+\tau_2) + \frac{1}{2}\sqrt{(\tau_1+\tau_2)^2} = 1+\tau_1+\tau_2.$$

$\hat{\lambda}_1$ is in fact the middle point of the interval

$$[1 + \frac{1}{2}(\tau_1+\tau_2) + \frac{1}{2}\sqrt{\tau_1^2+\tau_2^2},\ 1+\tau_1+\tau_2], \tag{4.12}$$

and

$$|\lambda_1 - \hat{\lambda}_1| \le \frac{1}{4}(\tau_1+\tau_2) - \frac{1}{4}\sqrt{\tau_1^2+\tau_2^2}.$$

Also, $\hat{\lambda}_2$ is the middle point of the interval

$$[1,\ 1 + \frac{1}{2}(\tau_1+\tau_2) - \frac{1}{2}\sqrt{\tau_1^2+\tau_2^2}]. \tag{4.13}$$

We can similarly show the bound for $|\lambda_2 - \hat{\lambda}_2|$. In particular, if $\tau_1 = \tau_2$, then

$$|\lambda_i - \hat{\lambda}_i| \le \frac{1}{2}(1 - \frac{\sqrt{2}}{2})\tau_1 < \frac{3}{16}\tau_1.$$

The case $|u_1| > |u_2|$ can be similarly shown and is omitted. ■

The implication of this proposition is as follows. If τ_1 and τ_2 are close to the tolerance τ but are not small enough to be ignored, we can further improve the accuracy of $\tilde{\lambda}_{1,2}$ by replacing them by $\hat{\lambda}_{1,2}$. The accuracy of the eigenvalues can be improved from τ_1 to either $\frac{3}{16}\tau_1$ or $\frac{3}{8}\tau_1$. In practice, this refined accuracy may be sufficient and then we avoid including additional low-rank updates in the DC scheme. In addition, the intervals give tighter bounds for $\lambda_{1,2}$ than the standard ones $[1+\tau_2,\ 1+\tau_1+\tau_2]$ and $[1,\ 1+\tau_1]$, respectively.

4.3.5 Effect of off-diagonal compression

As a test of the effect of off-diagonal compression, we form a 1000×1000 matrix by randomly (*Matlab* randn) selecting diagonal entires and randomly selecting off diagonal blocks of rank 2. Specifically, this is done by randomly generating basis vectors and orthonormalizing, setting singular values σ_1 and σ_2 as desired, and copying this to other off-diagonal block to preserve symmetry. The result is:

$$A = \begin{pmatrix} d_1 & (u_1, u_2)\mathrm{diag}(\sigma_1, \sigma_2)(v_1, v_2)^T \\ (u_1, u_2)\mathrm{diag}(\sigma_1, \sigma_2)(v_1, v_2)^T & d_2 \end{pmatrix}.$$

For various values of σ_1 and σ_2, we check the accuracy of the eigenvalues (element-wise) when σ_2 is set to 0. For each experiment we ran each value pair 20 times and average since there was decent variance from trial to trial. Though we note that the theoretical bounds showed later on were never violated within our numerical tests.

Table 4.2.
$\max_i\{\frac{\lambda_i - \tilde{\lambda_i}}{\eta_i}\}$ for various σ_1 and σ_2, illustrating that $|\lambda_i - \tilde{\lambda}_i| \leq \frac{\sigma_1\sigma_2}{\eta_i}$ in all cases.

σ_1/σ_2	$1e-5$	$1e-6$	$1e-7$	$1e-8$
$1e-1$	$8.56e-08$	$7.42e-09$	$5.38e-10$	$9.65e-10$
$1e-2$	$2.06e-08$	$2.03e-09$	$1.94e-10$	$3.47e-11$
$1e-3$	$5.46e-10$	$5.41e-11$	$4.41e-12$	$3.34e-13$
$1e-4$	$9.53e-11$	$3.34e-11$	$1.49e-12$	$8.43e-14$

4.3.6 Eigenspace contributions

Suppose we wish to compute the error of such an eigenspace contribution, $|\tilde{y}_i - y_i|$. If our HSS tree has only 2 levels (a 4×4 HSS matrix), then we can prove the following lemma which will be needed when proving tight error bounds.

Lemma: *Let y_i be an entry in the projection of the vector y at the root level of a block 4 HSS tree, then the error in this entry is bounded as follows:*$|\tilde{y}_i - y_i| \leq y_i \left(1 - \frac{\gamma_i}{\gamma_i - \epsilon_i}\right)$,

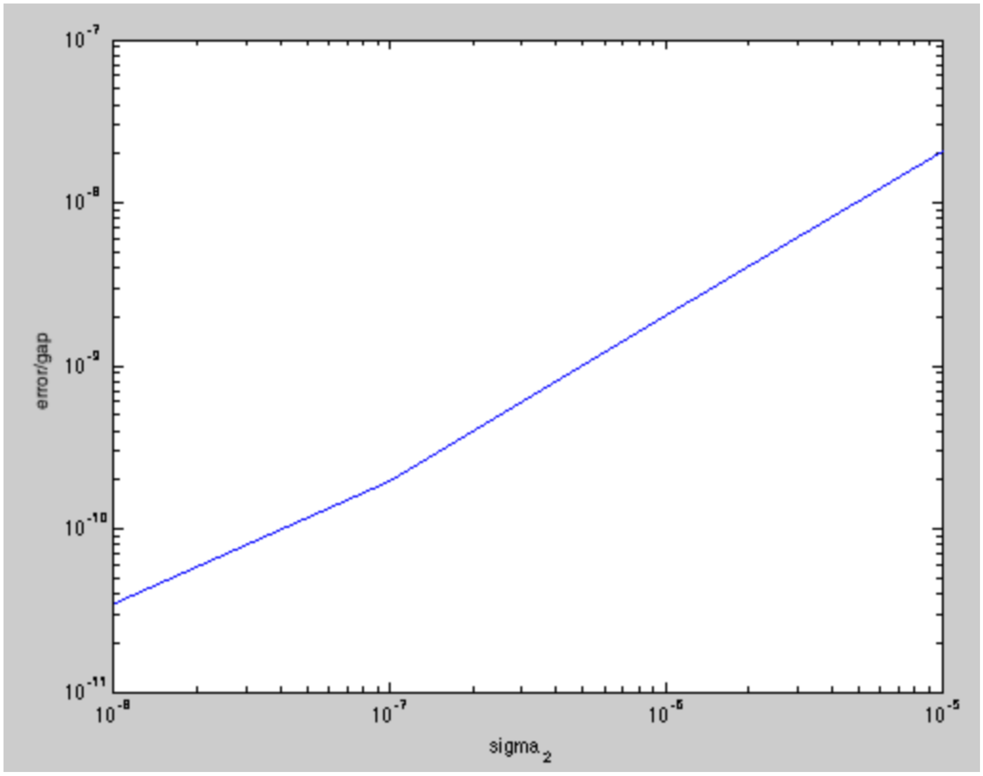


Fig. 4.2. holding σ_1 constant, we vary σ_2 and see how accuracy is affected when it is dropped.

where ϵ_i is the error of eigenvalue λ_i at the previous level and γ_i is the gap between eigenvalue λ_i and its closest neighbor.

Proof: Note that the eigenvector entries at the lower level theoretically take on the form:

$$Q(i,j) = \frac{v_i p_j}{d_i - \lambda_j}.$$

But there is an error induced by the off-diagonal compression. Suppose that the error of eigenvalue λ_i is bounded by ϵ_i, which in practice depends on both the compression tolerance and the gap between λ_i and its neighbors. So we are essentially projecting the rank-r updates by perturbed eigenvectors:

Fig. 4.3. effect of off-diagonal compression on eigenvalue accuracy

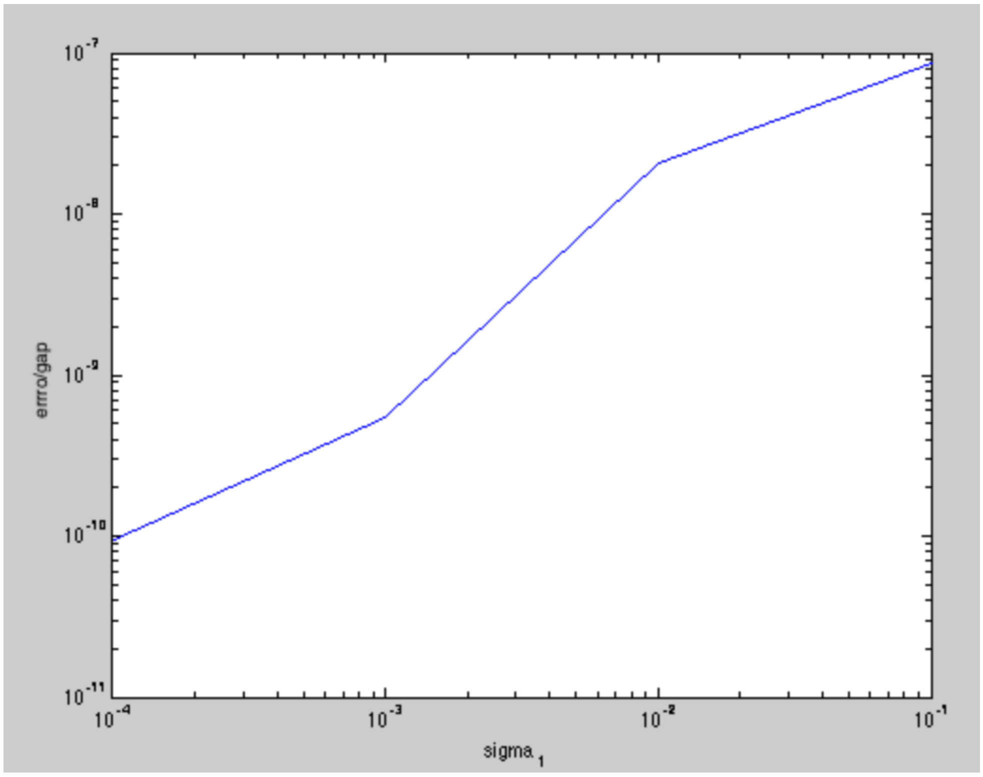


Fig. 4.4. holding σ_2 constant, we vary σ_1 and see how accuracy is affected when it is dropped.

$$Q(\tilde{i}, j) = \frac{v_i p_j}{d_i - \lambda_j \pm \epsilon_j}.$$

We now solve for the maximum perturbation of these projections. WLOG, assume that all entries of the vector v are positive and that the induced error ϵ_j is positive. In this way we can bound the the positive difference between the perturbed projection and the theoretical projection:

$$Q^T v(i) = \sum_{k=1}^{n} \frac{v_k^{(2)} v_k^{(1)} p_i}{d_k - \lambda_i - \epsilon_i}$$

$$\leq \sum_{k=1}^{n} \frac{v_k^{(2)} v_k^{(1)} p_i}{\gamma_i - \epsilon_i} = \frac{p_i}{\gamma_i - \epsilon_i} \sum_{k=1}^{n} v_k^{(2)} v_k^{(1)}.$$

But we need to consider the exact expression of p_i, or at least provide an upper bound for it in terms of the eigenvalue gap and the norm of the vector we are projecting:

$$p_i = \left(\sum_{k=1}^{n} \left(\frac{v_k^{(1)}}{d_k - \lambda_i} \right)^2 \right)^{-1/2} \leq \left(\sum_{k=1}^{n} \left(\frac{v_k^{(1)}}{\gamma_i} \right)^2 \right)^{-1/2}$$
$$= \gamma_i \left(\sum_{k=1}^{n} (v_k^{(1)})^2 \right)^{-1/2} = \frac{\gamma_i}{\|v^{(1)}\|_2}.$$

So we have that the bound expression for the perturbed projections is:

$$Q^T v(i) \leq \frac{\gamma_i}{\|v^{(1)}\|(\gamma_i - \epsilon_i)} \sum_{k=1}^{n} v_k^{(1)} v_k^{(2)},$$

and from a straightforward calculation it follows the error in these projections is bounded by:

$$|\tilde{y}_i - y_i| \leq y_i \left(1 - \frac{\gamma_i}{\gamma_i - \epsilon_i} \right) \quad \blacksquare$$

We note this gives us a lot of insight into the perturbation at each level.

$$A = \begin{pmatrix} D_1 & U_1 B_1 U_2^T & U_1 R_1 B_3 R_4^T U_4^T & U_1 R_1 B_3 R_5^T U_5^T \\ U_2 B_1^T U_1^T & D_2 & U_2 R_2 B_3 R_4^T U_4^T & U_2 R_2 B_3 R_5^T U_5^T \\ U_4 R_4 B_3^T R_1^T U_1^T & U_4 R_4 B_3^T R_2^T U_2^T & D_4 & U_4 B_4 U_5^T \\ U_5 R_5 B_3^T R_1^T U_1^T & U_5 R_5 B_3^T R_2^T U_2^T & U_5 B_4^T U_4^T & D_5 \end{pmatrix}$$
$$+ \left(\begin{array}{cc|cc} 0 & E_1 & & E_3 \\ E_1^T & 0 & & \\ \hline & & 0 & E_4 \\ E_3^T & & E_4^T & 0 \end{array} \right) \tag{4.14}$$

In future work, we will combine this result with the global theorems in [88] to come up with a more nuanced theory of eigenvalue perturbation analysis. In particular, with such a theory it may be possible to aggressively truncate off-diagonal blocks and still recover accurate eigenvalues. This could help stretch the utility of the algorithm to general matrix structures such as sparse (possibly with no discernible rank-pattern) matrices and the preconditioning of dense large unstructured matrices. We note that unlike the bounds in [88], the bounds in this section are not tight. That will be another important topic in future work, as understanding the worst case scenarios insofar as eigenvalue accuracy are concerned should should lots of light on this theory.

4.3.7 Conditioning of the secular equation

We now derive a secular equation for the form of the above tests. This is very important for practical applications; specifically in ones that have clustered eigenvalues. Many well-known eigenvalue algorithms are known to only perform well in cases when their spectra are well-separated. The divide-and-conquer flavor of algorithms presented in this thesis have been shown numerically to perform well in more pathological cases such as clustered eigenvalues, but the theoretical arguments in this subsection hope to shed a more rigorous light as to why this is the case. Analytically, we seek to find zeros of the following determinant function:

$$\det\left[D - I\lambda - \begin{pmatrix} u_1 & u_2 & 0 & 0 \\ 0 & 0 & u_1 u_2 & \end{pmatrix}\begin{pmatrix} 0 & v_1^T \\ 0 & v_2^T \\ v_1^T & 0 \\ v_2^T & 0 \end{pmatrix}\right]$$

Utilizing the product rule for determinants, we write:

$$= \det(D - I\lambda)\det\left[(I-(D-I\lambda)^{-1}\begin{pmatrix} u_1 & u_2 & 0 & 0 \\ 0 & 0 & u_1 u_2 & \end{pmatrix}\begin{pmatrix} 0 & v_1^T \\ 0 & v_2^T \\ v_1^T & 0 \\ v_2^T & 0 \end{pmatrix}\right].$$

But the first term is never nonzero if we have distinct eigenvalues and no nonzero components of our low-rank updates (and if we don't we can always deflate to that form), so we only need to consider finding the zeros of the following:

$$\det\left[(I-(D-I\lambda)^{-1}\begin{pmatrix} u_1 & u_2 & 0 & 0 \\ 0 & 0 & u_1 u_2 & \end{pmatrix}\begin{pmatrix} 0 & v_1^T \\ 0 & v_2^T \\ v_1^T & 0 \\ v_2^T & 0 \end{pmatrix}\right].$$

Now applying the Sylvester Determinant Theorem, one can show that this is equivalent to:

$$\det\left[\begin{pmatrix} 1 & 0 & \ldots \\ 0 & 1 & \ldots \\ -\sigma_1 \sum_{i=1}^{n/2} \frac{u_i^{(1)} v_i^{(1)}}{\lambda - d_i} & -\sqrt{\sigma_1\sigma_2}\sum_{i=1}^{n/2} \frac{u_i^{(2)} v_i^{(1)}}{\lambda - d_i} & \ldots \\ -\sqrt{\sigma_1\sigma_2}\sum_{i=1}^{n/2} \frac{u_i^{(1)} v_i^{(2)}}{\lambda - d_i} & -\sigma_2 \sum_{i=1}^{n/2} \frac{u_i^{(2)} v_i^{(2)}}{\lambda - d_i} & \ldots \end{pmatrix}\right].$$

At this point, one must simply compute the determinant explicitly by expansion and get the following secular equation:

$$f(\lambda) = 1 - \sigma_1^2 \left[\sum_{i=1}^{n/2} \frac{u_i^{(1)} v_i^{(1)}}{\lambda - d_i}\right]\left[\sum_{i=1}^{n/2} \frac{u_i^{(1)} v_i^{(1)}}{\lambda - d_{i+n/2}}\right] - \sigma_1\sigma_2 \left[\sum_{i=1}^{n/2} \frac{u_i^{(1)} v_i^{(2)}}{\lambda - d_i}\right]\left[\sum_{i=1}^{n/2} \frac{u_i^{(2)} v_i^{(1)}}{\lambda - d_{i+n/2}}\right]$$

$$-\sigma_1\sigma_2 \left[\sum_{i=1}^{n/2} \frac{u_i^{(2)} v_i^{(1)}}{\lambda - d_i}\right]\left[\sum_{i=1}^{n/2} \frac{u_i^{(1)} v_i^{(2)}}{\lambda - d_{i+n/2}}\right] - \sigma_2^2 \left[\sum_{i=1}^{n/2} \frac{u_i^{(2)} v_i^{(2)}}{\lambda - d_i}\right]\left[\sum_{i=1}^{n/2} \frac{u_i^{(2)} v_i^{(2)}}{\lambda - d_{i+n/2}}\right] \pm O(\frac{\sigma_1^2\sigma_2^2}{\eta}).$$

So if we drop σ_2, we get an element-wise error of:

$$|f(\lambda) - \tilde{f(\lambda)}| = \sigma_1\sigma_2 \left[\sum_{i=1}^{n/2} \frac{u_i^{(1)}v_i^{(2)}}{\lambda - d_i}\right] \left[\sum_{i=1}^{n/2} \frac{u_i^{(2)}v_i^{(1)}}{\lambda - d_{i+n/2}}\right]$$

$$+\sigma_1\sigma_2 \left[\sum_{i=1}^{n/2} \frac{u_i^{(2)}v_i^{(1)}}{\lambda - d_i}\right] \left[\sum_{i=1}^{n/2} \frac{u_i^{(1)}v_i^{(2)}}{\lambda - d_{i+n/2}}\right] + O(\frac{\sigma_2^2}{\eta})$$

From this it is clear that the error term is of the form $\frac{\sigma_1\sigma_2}{\eta_i}$.

Another way to see this is to examine the following bound.

$$\frac{\max_\lambda\{|f(\lambda) - \tilde{f(\lambda)}|\}}{\min_\lambda\{|f'(\lambda) - \tilde{f'(\lambda)}|\}},$$

This gives a similar asymptotically, though with slightly different constants. In future work, we will work to more rigorously understand this conditioning relationship and how it affects eigenvalue solutions in the context of global problems.

5. APPLICATIONS FOR THE STRUCTURED EIGENVALUE PROBLEM

In this chapter we give some examples of important applications of structured eigenvalue problems.

5.1 Functions of matrices

To begin, we note that functions of matrices are a simple example of how to apply our algorithms to solve various computational science and engineering, but with a profound impact. The well developed theory of functions of matrices allows us to apply the divide-and-conquer algorithm to a large class of problems. We note, as stated in [51], the following property:

Definition 5.1.1 *(matrix function via Jordan canonical form). Let f be defined on the spectrum of $A \in \mathbb{C}_{n\times n}$ and let A have the Jordan canonical form $A = AJZ^{-1}$. Then*

$$f(A) = Zf(j)Z^{-1} = Z diag(f(J_k))Z^{-1}.$$

Thus for any function of a symmetric matrix with a low-rank property, representable by a Taylor expansion, the function can be computed to high precision in nearly linear time and storage complexity, following the complexity analysis in [88]. As an example, consider the exponential of a Toeplitz matrix. This problem is of great interest in computational engineering and is still an active field of study. Yet the simple application of superfast divide-and-conquer is faster than available popular existing methods [9, 55, 75].

5.2 Spectral projectors

As another application, from the structured eigenvector matrix which can be computed in near-linear time, we obtain for no additional cost any spectral projector. Moreover, the projector is guaranteed to be orthogonal, which plays in important role in applications.

Let k be the number of eigenvectors required in our spectral projector. Then the cost of extracting the spectral projector from the superfast divide-and-conquer structured eigenvector matrix is:

$$\sum_{i=1}^{l_{\max}} 2^i \left(\frac{n\alpha}{2^i}\right) rk = O(rkn \log n).$$

This is comparable to other recent work on fast spectral projectors. A notable competitor algorithm is that of Kressner and Susnjara [56], which also utilizes the hierarchical structure of banded matrices. But instead of using eigenvector structure, they leverage the work of Nakatsukasa et al. [65–67] to utilize a variant the QDWH algorithm to compute the spectral projector. A very good survey of the use and justification for decay properties and hierarchical methods for banded and structured spectral projector can be found in [8]. However, the superfast divide-and-conquer eigensolver has two advantages as a spectral projector over other recent methods in that it will have guaranteed near machine precision orthogonality and it can be used for any matrix in the large class of symmetric HSS matrices; not just banded matrices.

5.3 Optimization

In this section we show the algorithms in this thesis and the one in [88] can be combined to greatly accelerate traditional barrier optimization methods. While extremely accurate and robust, traditional barrier methods are prohibitively slow for large problems and thus no longer commonly used in practice. However, our accelerated versions can be shown to be competitive with state-of-the art methods for quadratic programming problems, and likely would show even stronger results for more pathologically non-linear problems. Though we leave these pathological problems for future work. In particular, problems from the optimization area of semidefinite programming show a lot of promise to utilize our methods, and will certainly be revisited in future work.

The barrier method combines strict feasibility with the well-studied formalisms of unconstrained optimization algorithms. It generalizes well to pathologically nonlinear problems, and thus has lots of practical uses in applications. This method involves choosing a barrier function, which we denote here as $\varphi(x)$, that tends to ∞ as a constraint is nearly violated. We discuss the choice of barrier function below. Then, instead of optimizing our original objective function, we find the minima of a sequence of augmented problems:

$$\beta_\mu(x) = f(x) + \mu\varphi(x). \tag{5.1}$$

We start with μ as some moderately sized constant and let it go to zero. As shown in [36], this guarantees both convergence to the optimal solution, while maintaining strict feasibility throughout. There are, however, exhaustive possibilities on how to solve these unconstrained problems. In this thesis we focus on the standard Newton method with Wolfe line search, as presented in [36, 68], as well as how it can be accelerated using HSS algorithms.

It is noted in [36, 68] that several choices of barrier function $\varphi(x)$ are admissible. The two mentioned as common are $1/x$ and $\log(x)$. We note that the one used in our implementations was $\log(x)$.

First consider the special case of linear programming and with a logarithmic barrier function. We can write the augmented objective function, its gradient, and its Hessian cleanly and explicitly. Let $r = b - Ax$, and we use *Matlab* notation.

$$\beta_\mu(x) = \sum_{i=1}^{n} x_i - \mu \sum_{j=1}^{m} \log(b(j,1) - A(j,:)x) = \sum_{i=1}^{n} x_i - \mu \sum_{j=1}^{n} r(j,1) \tag{5.2}$$

$$\nabla(\beta_\mu(x))(i,1) = 1 + \mu \sum_{j=1}^{m} \frac{A(j,i)}{r(j,1)} \tag{5.3}$$

$$\nabla^2(\beta_\mu(x))(i,k) = \mu \sum_{j=1}^{m} \frac{A(j,i)A(j,k)}{r(j,1)^2} \tag{5.4}$$

If A can be approximated by a rank-k matrix, where k is very small compared to the size of A, then this Hessian can be written in terms of data-sparse generators, where the generators are formed using the Fast Multipole Method (FMM) with well-studied kernel function $1/x^2$. This is the primary reason $\log(x)$ was chosen. Further, it is commonly thought of as a reliable choice of barrier function [36, 68].

A similar argument can be made to show that general quadratic programming problems where the constraint matrices have a low-rank property also admit an HSS Hessian. This has been verified numerically. Moreover, our initial tests show us that several classes of more general non-linear optimization problems admit HSS Hessians. We will further investigate such problems in future work.

The bottleneck for HSS-Newton is still SVD, Hessian formation, and eigendecomposition, but they are considerably faster in this case. The complexity of the initial SVD performed is given in Section 2. Moreover, each FMM application is only $\mathcal{O}(m)$ so we can get all generators in $\mathcal{O}(mn)$. We do compute the band of the Hessian explicitly, but this can easily be shown to have negligible complexity for a small band. Moreover, as shown in [90], we can convert the banded plus semiseperable matrix to an HSS matrix via randomized sampling in $\mathcal{O}(r^2 \log n)$. Similarly, the entire eigendecompositon of an HSS matrix also only takes $\mathcal{O}(r^2 n \log n)$, when the algorithm in [88] used. Moreover, as an HSS matrix can be applied to a vector in $\mathcal{O}(rn \log^2 n)$, the matrix vector products involved in the finding the search direction are also quite fast.

Moreover, while this Hessian and its eigendecomposition will not be data sparse, the HSS approximation to the Hessian only requires $\mathcal{O}(n \log n)$ entries to be stored, which is comparable to the original sparse A. Moreover, as the eigenvectors of a symmetric HSS matrix can be shown to be HSS [88], and as the algorithm in [88] stores them in data-sparse form, the storage for the entire HSS-Newton barrier method

Algorithm 5 Accelerated Barrier Method for Linear or Non-Linear Optimization

```
 1: procedure HSS-Newton($A$, constraints) Output: optimal solution vector $x$
 2:     $A \approx U\Sigma V^T$ (pre-computation: HSS-SVD), with $U \in \mathbb{R}^{m\times k}$, $\Sigma \in \mathbb{R}^{k\times k,\text{diag}}$, $V \in \mathbb{R}^{n\times k}$, $k$ small
 3:     for $i = 1, 2, 3, ...10$ do $\mu = 10^{2-i}$
 4:         Solve for minimum of unconstrained problem: $\beta_\mu(x) = f(x) + \mu\varphi(x)$
 5:         while $\|\nabla(\beta_\mu)\| >$ tol do
 6:             Construct HSS approx. to Hessian $\tilde{H}$ (FMM, randomized sampling)
 7:             Compute eigendecomposition $\tilde{H} = Q\Lambda Q^T$
 8:             Modify $\Lambda$ as necessary to preserve positive definiteness
 9:             $p = Q\Lambda^{-1}Q^T(-\nabla(\beta_\mu))$
10:             Compute step length $\alpha$ with Wolfe Line Search
11:             $x \longleftarrow x + \alpha p$
12:         end while
13:     end for
14: end procedure
```

is $\mathcal{O}(rn \log n)$. This is a considerable savings from the non-accelerated version both in terms of operation count and storage.

We briefly mention two important features of our algorithms that guarantee convergence, as shown in [68]. In Newton's method, the Hessian matrix must be SPD to ensure we reach a global minimum (not a maximum or saddle point). But in practice, the matrix will have small negative eigenvalues at various stages of computation. In [68] it is shown that a very theoretically sound method of preserving the SPD form is taking an eigendecompositon of H, replacing all nonpositive eigenvalues with small positive ones, and proceeding. But this is rarely done in practice for the eigendecompositon is time consuming and may destroy the sparsity of H. However, in our case the run-time and storage for an HSS eigendecompositon is quasilinear, so we can use this method without difficulty. Our line search is implemented with the Strong Wolfe conditions.

We cannot guarantee accuracy of our HSS-Newton method. That is because in this method we are not really solving the stated problem but a *nearby problem with nicer properties.* Moreover, the question of just how “nearby” we are is completely problem-dependent (based on the decay rate of the singular values). We tested 40 full rank LP problems from the UF collection, and only in 3 cases (80bau3b being the only one in representative results shown below) were we successful, where we say the implementation is “successful” if it satisfies the following 3 properties:

- algorithm converges in time less than a standard Newton implementation
- function value $f(x)$ is of correct order of magnitude with ≥ 2 digits correct
- x lies in feasible region of original problem, not just of nearby problem

In the cases where HSS-Newton was not successful, our implementation re-runs with taking a full-SVD. In such cases, the code gives the correct solution but the total run time is much slower than a standard Newton. In conclusion, HSS-Newton is a very fascinating idea theoretically, and in certain situations it can accelerate Newton's method, but it it does not seem to be a reliable strategy for general LP problems.

Finally, we consider a specific example of HSS-optimization where our method has a clear advantage over existing methods. We examine semidefinite programs with constraints derived from the Kalman-Yakubovich-Popov lemma, commonly referred to as KYP-SDP optimization problems [87]. These problems take the form:

$$\begin{aligned} &\min \quad c^T x \\ &\text{s.t.} \quad K_i^H(\varphi \otimes P_i)K_i + M_i(x) \succeq 0, i = 1, ..., L \end{aligned}$$

A traditional method for these problems is an interior-point method such as in [86,87]. But it was noted in [59] that this problem has rank structure, and semiseparable matrix structure was used to accelerate the standard algorithm faster than the

Table 5.1.
comparison of Newton's method and HSS-Newton for some linear programming problems

Newton	Size	Run Time	Optimal Solution	% error
itest6	11x17	0.012s	.0000	-
chemcom	288x744	8.727s	825.0000	0.00%
80bau3b	2262x12061	363.650s	700.0000	0.00%
HSS-Newton	Size	Run Time	Optimal Solution	% error
itest6	11x17	3.957s	0.0035	-
chemcom	288x744	12.154s	825.0453	0.00%
80bau3b	2262x12061	29.607s	709.3344	1.33%

standard $O(n^6)$ complexity. However, if these algorithms are extended in a straightforward way using more sophisticated rank structures and more modern implementations such as superfast DC, the algorithm can be accelerated all the way to nearly $O(n^3)$ complexity, a drastic improvement of almost three orders of magnitude. In the following section, this claim will be illustrated with numerical experiments. Moreover, many other classes of semidefinite programs, such as those listed in [86] also can exploit rank-structure to dramatically accelerate convergence. This will be explored more extensively in future work.

6. FUTURE WORK

In the future I plan to continue to make several contributions to this research program. I will continue to work on eigenvalue algorithms, eigenvalues analysis, and eigenvalue applications, as well as continue related work in rank-structured optimization.

For algorithms, there is more work that can be done for an effective multi-rank update. The inverse algorithm wherein has some robustness issues that need to be addressed. Moreover, this algorithm can be extended for multi-rank updates for sparse eigenvalue problems, multi-rank singular value decomposition updates, and multi-rank generalized eigenvalue decomposition updates. As far as global eigenvalue algorithms, work can be done to extend this work to high-dimensional problems, non-symmetric eigenvalue problems, and even non-linear problems.

For analysis, much more work can be done doing rudimentary work on accuracy and convergence properties of multi-rank updates to the symmetric eigenvalue problem. Methods will likely involve looking at individual sup-spaces in more detail than this thesis. This will lead to tighter and more useful bounds for global eigenvalue algorithm accuracy, and should allow us to consider more pathological problems such as those with clustered spectra.

For applications, there is a lot of work that can be done in computational physics applications; such as quantum mechanics, statistical mechanics, numerical relativity, astrophysics, scattering theory, and plasma physics. There is also further work that can be done of general partial differential equation solvers. The work in this thesis can be extended to challenging national security problems like passive aperture radar and cyber-physical systems. Finally, it is our hope to continue to work on the promising application of medical imaging discussed in the introduction of this thesis.

In terms of optimization, this thesis laid the groundwork for a promising new flavor of algorithms, and we plan to explore this channel in depth. Initial work will be on linear programming, quadratic programming, integer linear programming, and geolocation problems. Down the road, as the theory becomes more well understood, it will be possible to do work such as structural dynamics, matrix equations, sparse principal component analysis, mixed integer quadratic programming, linear matrix inequalities, and non-linear programming (via automatic differentiation).

The author of this work is very excited for a long and challenging career as a research mathematician!

REFERENCES

REFERENCES

[1] S. Ambikasaran and E. Darve, *The inverse fast multipole method*, arXiv preprint arXiv:1407.1572 (2014).

[2] J. Anderson, *A Secular Equation for the Eigenvalues of a Diagonal Matrix Perturbation*, Linear Algebra and Its Applications 246 (1996): 49-70.

[3] S. ARORA, E. HAZAN, AND S. KALE, *A Fast Random Sampling Algorithm for Sparsifying Matrices, Approximation, Randomization, and Combinatorial Optimization*. Algorithms and Techniques, Springer Berlin, 2006, pp. 272-279.

[4] H. Bagci, J. E. Pasciak, and K. Y. Sirenko, *A convergence analysis for a sweeping preconditioner for block tridiagonal systems of linear equations*, Numer. Linear Algebra Appl., 22 (2015), pp. 371–392.

[5] J. Barnes AND P. Hut, *A hierarchical $O(N \log N)$ force-calculation algorithm*, Nature. 324 (1986), pp. 446-449.

[6] R. BEATSON AND L. GREENGARD, *A short course on fast multipole methods*, Wavelets, Multilevel Methods and Elliptic PDEs, Oxford University Press, (1997), pp. 1-37.

[7] C. BISCHOF, X. SUN, A. TSAO, AND T. TURNBULL, (1994). *A study of the Invariant Sub- space Decomposition Algorithm for banded symmetric matrices*, Proceedings of the Fifth SIAM Conference on Applied Linear Algebra.

[8] M. Benzi, P. Boito, and N. Razous, *Decay properties of spectral projectors with applications to electronic structure*, SIAM Rev., 55 (2013), pp. 3-64.

[9] D. A. Bini, S. Massei, and B. Meini, *On Functions of quasi Toeplitz matrices*, Sbornik: Mathematics 208.11 (2017): 1628.

[10] C. BISCHOF, S. HUSS-LEDERMAN, X. SUN, A. TSAO, AND T. TURNBULL, *Parallel Studies of the Invariant Subspace Decomposition Approach for Banded Symmetric Matrices*, PPSC (1995) pp. 516-521.

[11] S. BOUCHERON, G. LUGOSI, AND P. MASSART, *Concentration inequalities using the entropy method*, Ann. Probab. 31 (2003), pp. 1583-1614.

[12] M. Brand *Fast low-rank modifications of the thin singular value decomposition*, Linear Algebra. Appl. 415 (2006): pp. 20-30.

[13] J. R. Bunch, C. P. Nielson, and D. C. Sorenson, *Rank-one modification of the symmetric eigenproblem*, Numer. Math., 41 (1978), pp. 31-48.

[14] D. Cai and J. Xia, *A stable and efficient matrix version of the fast multipole method*, preprint to be submitted, 2015, http://www.math.purdue.edu/~xiaj/work/fmm1d.pdf.

[15] J. Carrier, L. Greengard, and V. Rokhlin. *A fast adaptive multipole algorithm for particle simulations*, SIAM J. Sci. Stat. Comput., 9 (1988), pp. 669-686.

[16] S. Cauley, Y. Xi, B. Bilgic, J. Xia, E. Adalsteinsson, V. Balakrishnan, L. Wald, and K. Setsompop, *Fast reconstruction for multi-channel compressed sensing using a hierarchically semiseparable solver*, Magn. Reson. Med., 73 (2015), pp. 1034-1040.

[17] W.-M. Chan and A. George, *A linear time implementation of the reverse Cuthill-McKee algorithm* BIT Num. Math. 20 (1980), pp. 8-14.

[18] S. Chandrasekaran, P. Dewilde, M. Gu, T. Pals, X. Sun, A. H. van der Veen, AND D. White, *Some fast algorithms for sequentially semiseparable representations*. SIAM Journal on Matrix Analysis and Applications, 27 (2005), pp. 341-364.

[19] S. Chandrasekaran and M. Gu, *A divide-and-conquer algorithm for the eigendecomposition of symmetric block diagonal plus semiseparable matrices*, Numer. Math., 96 (2004), pp. 723–731

[20] S. CHATTERJEE, *Stein's method for concentration inequalities*, Probab. Theory Relat. Fields 138 (2007), pp. 305-321.

[21] J. J. M. CUPPEN, *A divide and conquer method for the symmetric tridiagonal eigenproblem*, Numer. Math., 36 (1981), pp. 177-195.

[22] J. Demmel, *Applied Numerical Linear Algebra*, SIAM, Philadelphia, PA, 1997.

[23] P. DRINEAS, M. W. MAHONEY, AND S. MUTHUKRISHNAN, *Relative-Error CUR Matrix Decompositions*, SIAM J. Matrix Anal. Appl. 30 (2008), pp. 844-881.

[24] J. J. DONGARRA AND D. C. SORENSEN, *A fully parallel algorithm for the symmetric eigenvalue problem*, SIAM J. Sci. Stat. Comput. 8 (1987), pp. 139-154.

[25] J. Dubinski *A Parallel Tree Code*, New Astronomy. 1 (1996), pp. 133-147.

[26] I. S. Duff, and J. K. Reid, *The multifrontal solution of indefinite sparse symmetric linear*, ACM Transactions on Mathematical Software (TOMS) 9.3 (1983), pp. 302-325.

[27] R. ECKHARDT, *Stan ulam, john von neumann, and the monte carlo method*, Los Alamos Science 15 (1987), pp. 131-136.

[28] L. Elsner, A. Fasse, and E. Langmann, *A divide-and-conquer method for the tridiagonal generalized eigenvalue problem*, J. Comput. Appl. Math. 86 (1997), pp. 141-148.

[29] E. GALLOPOULOS, B. PHILLIPE, A. SAMEH, *Parallelism in Matrix Computations*, Springer, Scientific Computation Series, 2015.

[30] A. George, *Nested dissection of a regular finite element mesh*, SIAM Journal on Numerical Analysis 10.2 (1973), pp. 345-363.

[31] I. Gohberg, T. Kailath, and V. Olshevsky, *Fast Gaussian elimination with partial pivoting for matrices with displacement strcutres*, Math. Comp., 64 (1995), pp. 1557–1576.

[32] G. H. GOLUB, Some modified matrix eigenvalue problems, SIAM Rev. 15 (1973), pp. 318-334.

[33] G. Golub AND C. V. Loan, *Matrix Computations*, 4th ed., The Johns Hopkins University Press, Baltimore, MD, 2013.

[34] L. Grasedyck and W. Hackbusch, *Construction and arithmetics of $\mathcal{H}$-matrices*, Computing, 70 (2003), pp. 295–334.

[35] L. GREENGARD AND V. ROKHLIN. *A fast algorithm for particle simulations*, J. Comp. Phys., 73 (1987), pp. 325-348.

[36] I. Griva, G. Nash, and A. Sofer, *Linear and Nonlinear Optimization*, Philadelphia: Society for Industrial and Applied Mathematics, 2009.

[37] M. Gu, *Stable and efficient algorithms for structured linear equations*, SIAM J. Matrix Anal. Appl., 19 (1998), pp. 279–306.

[38] M. GU, *Subspace iteration randomization and singular value problems*, SIAM Journal on Scientific Computing 37 (2015), pp. 1139-1173.

[39] M. GU AND S. C. EISENSTAT, *A divide-and-conquer algorithm for the symmetric tridiagonal eigenproblem*, SIAM J. Matrix Anal. Appl., 16 (1995), pp. 79-92.

[40] M. Gu and S. C. Eisenstat, *Efficient algorithms for computing a strong rank-revealing QR factorization*, SIAM Journal on Scientific Computing 17.4 (1996), pp. 848-869.

[41] M. Gu, S. C. Eisenstat, *Downdating the singular value decomposition*, SIAM J. Matrix Anal. Appl., 16 (1995) pp. 793-810.

[42] W. Hackbusch, *Multi-grid methods and applications*, Springer Science & Business Media, 1985.

[43] W. Hackbusch, *A sparse matrix arithmetic based on $\mathcal{H}$-matrices. Part I: Introduction to $\mathcal{H}$-matrices*, Computing, 62 (1999), pp. 89-108.

[44] W. Hackbusch, *Hierarchische Matrizen: Algorithmen und Analysis*, Springer, Berlin, 2009.

[45] W. Hackbusch and S. Börm, *Data-sparse approximation by adaptive $\mathcal{H}^2-$matrices*, Computing, 69 (2002), pp. 1-35.

[46] W. Hackbusch and B. N. Khoromskij, *A sparse $\mathcal{H}$-matrix arithmetic. Part II: Application to multi-dimensional problems*, Computing, 64 (2000), pp. 21-47.

[47] W. Hackbusch, B. Khoromskij, and S. Sauter, On $\mathcal{H}^2$-matrices, in Lectures on Applied Mathematics, H. Bungartz, R. H. W. Hoppe, and C. Zenger, eds., Springer, Berlin, 2000, pp. 9-29.

[48] W. W. Hager, *Perturbations in eigenvalues*, Linear Algebra Appl., 42 (1982), pp. 39–55.

[49] N. HALKO, P. G. MARTINSSON, AND J. A. TROPP, *Finding Structure with Randomness : Probabilistic Algorithms for Constructing Approximate Matrix Decompositions*, SIAM Rev. 53 (2011), 217-288.

[50] G. Heineg, *Inversion of generalized Cauchy matrices and other classes of structured matrices*, Linear Algebra for Signal Processing, IMA Vol. Math. Appl. 69, Springer, New York, 1995, pp. 63–81.

[51] N. Higham, *Functions of matrices: theory and computation*, SIAM, Philadelphia, PA, 2008.

[52] J.T. HOLODNAK, I.C.F. IPSEN AND T. WENTWORTH, *Conditioning of Leverage Scores and Computation by QR Decomposition*, SIAM J. Matrix Anal. Appl., vol. 36 (2015), pp. 1143-1163.

[53] I. C. F. Ipsen and B. Nadler, *Refined perturbation bounds for eigenvalues of Hermitian and non-Hermitian matrices*, SIAM J. Matrix Anal. Appl., 31 (2009), pp. 40–53.

[54] E.-R. Jiang, *Perturbation in eigenvalues of a symmetric tridiagonal matrix*, Linear Algebra Appl., 399 (2005), pp. 91–107.

[55] D. Kressner and R. Luce, *Fast computation of the matrix exponential for a Toeplitz matrix*, arXiv preprint arXiv:1607.01733 (2016).

[56] D. Kressner and A. Susnjara, *Fast computation of spectral projectors of banded matrices*, arXiv preprint arXiv:1608.01164 (2016).

[57] W. Kahan, *When to neglect off-diagonal elements of symmetric tridiagonal matrices*, Technical Report CS42, Computer Science Department, Stanford University, July 1966.

[58] C. L. Lawson and R. J. Hanson, *Solving Least Squares Problems.* Englewood Cliffs, NJ, Prentice Hall, 1974.

[59] Z. Liu and L. Vandenberghe, *Low-rank structure in semidefinite programs derived from the KYP lemma*, Decisions and Contro, 2007 46th IEEE Conference on IEEE, 2007.

[60] M. MAHONEY, *Combinatorial scientific computing, ch. Algorithmic and Statistical Perspectives on Large-Scale Data Analysis*, Chapman and Hall/CRC, 2012.

[61] P. G. Martinsson, V. Rokhlin, and M. Tygert, *A fast algorithm for the inversion of general Toeplitz matrices*, Comput. Math. Appl. 50 (2005), pp. 742–752.

[62] T. Melzer, *SVD and its application to generalized eigenvalue problems*, Vienna University of Technology, 2004.

[63] T. Kailath, S. Kung, and M. Morf, *Displacement ranks of matrices and linear equations*, J. Math. Anal. Appl., 68 (1979), pp. 395–407.

[64] R.-C. LI, *Solving secular equations stably and efficiently*, Technical Report UT-CS-94-260, University of Tennessee, Knoxville, TN, Nov. 1994. LAPACK Working Note No. 89.

[65] Y. Nakatsukasa, Z. Bai, and F. Gygi, *Optimizing Halley's itereation for computing the matrix polar deomposition,* SIAM J. Matrix Anal. Appl., 31 (2010), pp. 2700-2720.

[66] Y. Nakatsukasa and R. W. Freudn, *Computing fundamental matrix decompositions accurately via the matrix sign function in two iterations: The power of Zolotarev's functions*, SIAM Rev. 58 (2016), pp. 461-493.

[67] Y. Nakatsukasa and N. J. Higham, *Stable and efficient spectral divide and conquer algorithms for the symmetric eigenvalue decompositions and the SVD*, SIAM J. Sci. Comput., 35 (2013), pp. A1325-A1349..

[68] J. Nocedal and S. Wright, *Numerical Optimization*, New York: Springer, 2006.

[69] C. C. PAIGE, *Eigenvalues of perturbed hermitian matrices*, Linear Algebra Appl., 8 (1974), pp. 1-10.

[70] V. Y. Pan, *On computations with dense structured matrices*, Math. Comp., 55 (1990), pp. 179–190.

[71] V. Y. Pan, *Trasnformations of matrix structures work again*, Linear Algebra Appl., 465 (2015), pp. 107–138.

[72] B. N. Parlett, *The Symmetric Eigenvalue Problem*, SIAM, PA, 1998.

[73] V. ROKHLIN AND M. TYGERT, *A fast randomized algorithm for overdetermined linear least-squares regression*, Proc. Natl. Acad. Sci. USA 105 (2008), pp. 13212-13217.

[74] F.H. Rouet, X. S. Li, P. Ghysels, AND A. Napov,*A distributed-memory package for dense hierarchically semi-separable matrix computations using randomization.* ACM Transactions on Mathematical Software (TOMS), 42 (2016), 27.

[75] A. Sakhnovich, *Toeplitz matrices with an exponential growth of entries and the first Szego limit theorem*, J. of Func.Anal. 171 (2000), pp. 449-482.

[76] J. Shen, T, Tang, and L. Wang, *Spectral methods algorithms, analysis and applications*, Springer Science and Business Media, 2011.

[77] D. Spielman and S.-H.Teng, *Disk packings and planar separators*, Proceedings of the twelfth annual symposium on computational geometry, ACM, 1996.

[78] D. C. Sorensen and P. T. P. Tang, *On the orthoganality of eigenvectors computed by divide-and-conquer techniques*, SIAM J. Numer. Anal., 28 (1991), pp. 1752-1775.

[79] P. Starr, *On the Numerical Solution of One-Dimensional Integral and Differential Equations*, PhD Thesis, Yale University, New Haven, CT, 1992.

[80] J. .J. SYLVESTER, *A demonstration of the theorem that every homogeneous quadratic polynomial is reducible by real orthogonal substitutions to the form of a sum of positive and negative squares*, Philos. Mag., IV (1852), pp. 138-142.

[81] F. TISSEUR AND J. DONGARRA, *A parallel divide and conquer algorithm for the symmetric eigenvalue problem on distributed memory architectures*, SIAM Journal on Scientific Com- puting 20 (1999), pp. 2223-2236.

[82] W. F. Trench, *Numerical solution of the eigenvalue problem for Hermitian Toeplitz matrices*, SIAM J. Matrix Anal. Appl., 10 (1989), pp. 135–156.

[83] E. Tyrtyshnikov, *Mosaic-skeleton approximations*, Calcolo 33.1-2 (1996), pp. 47-57.

[84] E. Tyrtyshnikov, *Incomplete cross approximation in the mosaic-skeleton method*, Computing 64.4 (2000), pp. 367-380.

[85] R. Vandebril, G. Golub, and M. Van Barel, *A quasi-separable approach to solve the symmetric definite tridiagonal generalized eigenvalue problem*, SIAM J. Matrix Anal. Appl., 31 (2009), pp. 154-174.

[86] L. Vandenberghe and S. Boyd, *Semidefinite programming*, SIAM review 38.1 (1996): 49-95.

[87] L. Vandenberghe, V Balakrishnan, R. Wallin, A. Hansson, and T. Roh, *Interior-point algorithms for semidefinite programming problems derived from the KYP lemma*, Positive polynomials in control. Springer Berlin Heidelberg, 2005. 195-238.

[88] J. Vogel, J. Xia, S. Cauley, and V. Balakrishnan, *A superfast divide-and-conquer method and approximation accuracy for structured eigenvalue solutions*, SIAM J. on Scientific Computing, 38 (2016), pp. A1358-1382. Copyright 2016 Society of Industrial and Applied Mathematics. Reprinted with permission. All rights reserved.

[89] Y. Xi, J. Xia, S. Cauley, and V. Balakrishnan, *Superfast and stable structured solvers for Toeplitz least squares via randomized sampling*, SIAM J. Matrix Anal. Appl., 35 (2014), pp. 44–72.

[90] Y. XI, J. XIA, AND R. CHAN, *A fast randomized eigensolver with structured LDL factorization update*, SIAM J. Matrix Anal. Appl., 35 (2014), pp. 974-996.

[91] J. XIA, *Fast direct solvers for structured linear systems of equations*, Ph.D. Thesis ,University of California, Berkeley, 2006.

[92] J. XIA, *On the complexity of some hierarchical structured matrix algorithms*, SIAM J. Matrix Anal., 33 (2012), pp. 388-410.

[93] J. Xia, *Efficient structured mutlifrontal factorization for general large sparse matrices*, SIAM J. Sci. Comput., 35 (2013), pp. 832-860

[94] J. Xia, *Multi-layer hierarchical structures and factorizations*, SIAM J. Matrix Anal. Appl., submitted, 2016, Purdue CCAM Report CCAM-2016-6.

[95] J. XIA, S. CHANDRASEKARAN, M. GU, AND X. S. LI, *Fast algorithms for hierarchically semiseparable matrices*, Numer. Linear Algebra Appl., 17 (2010), pp. 953-976.

[96] J. Xia, Y. Xi, S. Cauley, and V. Balakrishnan, *Fast sparse selected inversion*, SIAM J. Matrix Anal. Appl, 2015.

[97] J. XIA, Y. XI, AND M. GU, *A superfast structured solver for Toeplitz linear systems via randomized sampling*, SIAM J. Matrix Anal. Appl., 33 (2012), pp. 837?858.

[98] Z. Xin, J. Xia, M. V. de Hoop, S. Cauley, and V. Balakrishnan, *A distributed-memory randomized structured multifrontal method for sparse direct solutions*, SIAM J. Sci. Comput., 39 (2017), pp. C292-C318.

[99] 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 (2016), Article 21.

[100] D. S. WATKINS, *The Matrix Eigenvalue Problem: QR and Krylov Subspace Methods*, SIAM, Philadelphia, PA, 2007.

[101] J. H. WILKINSON, *The algebraic eigenvalue problem*, Clarendon Press, Oxford, 1965.

[102] H. WEYL Inequalities between the Two Kinds of Eigenvalues of a Linear Transformation, Proc Natl Acad Sci U S A. 7 (1949), pp. 408-411.

[103] Q. Ye, *Relative perturbation bounds for eigenvalues of symmetric positive definite diagonally dominant matrices*, SIAM J. Matrix Anal. Appl., 31 (2009), pp. 11–17.

RE: permission request for doctoral dissertation

Kelly Thomas <Thomas@siam.org>

Wed 9/19/2018 4:54 PM

Inbox

To:James P Vogel <vogel13@purdue.edu>;

Dear Mr. Vogel:

SIAM is happy to give permission to reprint material from the article mentioned below. In the credit line, please cite the complete bibliographic information for the original article and include the wording “Copyright ©2016 Society for Industrial and Applied Mathematics. Reprinted with permission. All rights reserved.”

Sincerely,

Kelly Thomas
Managing Editor, SIAM
thomas@siam.org

From: James P Vogel [mailto:vogel13@purdue.edu]
Sent: Wednesday, September 19, 2018 4:26 PM
To: Kelly Thomas <Thomas@siam.org>
Subject: permission request for doctoral dissertation

Hello Ms. Thomas,

I am writing to request permission to use material from the SIAM publication

James Vogel, Jianlin Xia, Stephen Cauley, Venkataramanan Balakrishnan, *Superfast Divide-and-Conquer Method and Perturbation Analysis for Structured Eigenvalue Solutions*, SIAM Journal on Scientific Computing, 38 (2016), pp. A1358-A1382

in my doctoral thesis at Purdue University. I am an author of the above article and SIAM holds the copyright. I plan to reuse text, figures, and tables from the above mentioned paper. My university dissertations are submitted to ProQuest. ProQuest may supply single copies of the dissertation on demand. If consent is given to use the above material, I will acknowledge the SIAM copyright for the above article in my dissertation in accordance with university guidelines.

Best,
James Vogel

VITA

VITA

Jimmy Vogel was born in San Diego, California, USA in 1991. In 1995 his family moved to St. Paul, Minnesota, where he lived until 2009. In May of 2013 he received a B.S. in Mathematics and Physics from the University of Michigan with minors in Ancient Roman History and Clarinet Performance. He then enrolled in Purdue University's Mathematics graduate program where he joined the research group of Prof. Jianlin Xia. In addition to his many research accomplishments, Jimmy served as Senator and Treasurer of Purdue's Graduate Student Government (PGSG), Grad Rep. for the Math Department, Vice-President for Purdue's chapter of the Society for Industrial and Applied Mathematics, and organized several local conferences and seminars. In addition to being an aspiring applied mathematician, Jimmy is also a virtuoso clarinet player, an above average soccer player, one of Purdue's current Top 5 table tennis players, and a notable chocolate cake enthusiast. His two most distinguished awards to date are the PGSG's "Above and Beyond" Award in 2017 and the Purdue College of Science's Cagiantas Dissertation Fellowship in 2017-2018.