Title
Fast and Stable Low-Rank Symmetric Eigen-Update

Author
Liang, Ruochen

Publication Date
2018

Peer reviewed|Thesis/dissertation

Fast and Stable Low-Rank Symmetric Eigen-Update

by

Ruochen Liang

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Applied Mathematics

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Ming Gu, Chair
Professor John Strain
Professor Peter Bartlett

Summer 2018

Fast and Stable Low-Rank Symmetric Eigen-Update

Abstract

Fast and Stable Low-Rank Symmetric Eigen-Update

by

Ruochen Liang

Doctor of Philosophy in Applied Mathematics

University of California, Berkeley

Professor Ming Gu, Chair

Updating the eigensystem of modified symmetric matrices is an important task arising from certain fields of applications. The core of the problem is computing the eigenvalues and orthogonal eigenvectors of a diagonal matrix with symmetric low rank modifications, *i.e.* $D+UHU^T$. The eigenproblem of this type of matrix has long been studied since Golub et.al [45] proposed theoretical considerations about it. Currently, there exist methods to compute eigenvalues accurately, but the difficulty remains in numerical stability and orthogonality of computed eigenvectors.

The main contribution of this thesis is a new method to compute all the eigenvalues and eigenvectors of a real diagonal matrix with a symmetric low rank perturbation. The algorithm computes an orthogonal matrix $Q = [q_1, q_2, \ldots, q_n]$ and a diagonal matrix $\Lambda = diag\{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ such that $AQ = Q\Lambda$. Here the matrix $A = D + UHU^T$ has the special structure that $D \in \mathbb{R}^{n\times n}$ is a diagonal matrix, $U \in \mathbb{R}^{n\times r}$ is a column orthorgonal matrix and $H \in \mathbb{R}^{r\times r}$ is a symmetric matrix. n is the dimension of A and $r << n$ is the rank of the low-rank perturbation.

Aside from solving the eigensystem update problem mentioned above, our proposed method can also be used in the divide and conquer eigenvalue algorithm. Cuppen's divide and conquer algorithm [16] solves a rank-one update of eigensystem in its merge step for a symmetric tri-diagonal matrix. A symmetric banded matrix will require solving the eigensystem of a low-rank perturbed diagonal matrix, *i.e.*, $D + UHU^T$. Efficient solution to this problem in the merge step can potentially enable application of divide and conquer algorithm directly on symmetric banded matrix.

In our proposed algorithm, eigenpairs are mostly computed by Rayleigh Quotient Iteration safe-guarded with bisection, with each eigenpair requiring $O(nr^2)$ flops to compute. Hence the overall computational complexity for our algorithm is $O(n^2r^2)$. This is an appealing quadratic algorithm since r is usually considered a small constant relative to n. To ensure numerical stability, eigenvectors corresponding to eigenvalue clusters are computed through a special *orthogonal deflation* method that completely avoids re-orthogonalization. In case of tight clusters, extended precision arithmetic is

used for eigenvectors corresponding to close eigenvalues. We present both theoretical analysis and numerical results to support our claim that the proposed algorithm is numerically stable.

To my parents and friends, for their consistent love and support.

Contents

Acknowledgments

The author would like to express his extreme gratitude to Professor Ming Gu for making worthwhile the experience of working on this thesis. The author has benefited a lot from Professor Ming Gu's expertise and vision in conducting mathematical research as well as his patience in presenting the work. The author is also grateful to Professor Ming Gu for introducing him to the field of numerical linear algebra. Professor Gu's constant support, numerous suggestions and careful reading have played an important role in the development of this thesis. Besides, helpful office discussions have greatly shaped the work and sparked ideas.

The author is also grateful to many friends and fellow Ph.D students who have made life at Berkeley a fun and enriching experience.

Chapter 1

Introduction

In this thesis we present a new algorithm to compute all eigenvalues and eigenvectors of a diagonal matrix with a symmetric low-ran modification. The algorithm computes an orthogonal matrix $Q = [q_1, q_2, \ldots, q_n]$ and a diagonal matrix $\Lambda = diag\{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ such that $AQ = Q\Lambda$. Here the matrix $A = D + UHU^T$ has the special structure that $D \in \mathbb{R}^{n\times n}$ is a diagonal matrix, $U \in \mathbb{R}^{n\times r}$ is a column orthorgonal matrix and $H \in \mathbb{R}^{r\times r}$ is a symmetric full rank matrix. n is the size of the eigenvalue problem and $r << n$ is the rank of the low-rank perturbation.

The main advance is in being able to compute numerically orthogonal eigenvectors without the need of Gram-Schmidt or any other similar technique to explicitly re-orthogonalize eigenvectors. Existing methods for such low rank update usually treat it as a sequence of rank-one update and result in expensive accumulation of eigenvectors. Our new algorithm is the result of a combination of methods that enable us to compute in $O(n^2)$ time, accurate and numerically orthogonal eigenvectors. We believe that our method can be further applied in some other numerical linear algebra problems.

1.1 Objective

Before sketching the outline of the thesis, we first list some goals of our algorithm:

1. $O(n^2)$ complexity. We aim to achieve the minimum complexity in computing all eigenvectors.

2. An accurate algorithm. With the limit of finite precision arithmetic, we cannot hope to compute true eigenvalues and orthogonal eigenvectors. A realistic goal is to compute approximate eigenpairs $(\hat{\lambda}_i, \hat{q}_i), i = 1, 2, \ldots, n$ such that:

   - The residual norms are relatively small:

$$||A\hat{q}_i - \hat{\lambda}_i\hat{q}_i|| = O(n\epsilon||A||). \tag{1.1}$$

- The computed eigenvectors are numerically orthogonal:

$$|\hat{q}_i^T \hat{q}_j| = O(n\epsilon), \quad \forall i \neq j, \tag{1.2}$$

where ϵ is a given machine precision. We will discuss how these properties relate to backward stability of the eigen-decomposition in later chapters.

3. An adaptive algorithm that allows the computation of k largest/smallest eigenvalues and the corresponding eigenvectors at a reduced cost.

Our algorithm presented in chapter 3.4 has achieved the first two goals. It's overall complexity is $O(n^2r^2)$ where r is the rank of the low rank modification. The accuracy requirements are further analyzed in chapter 6. Due to the nature of our bracketing algorithm, it is difficult to control the set of computed eigenvalues.

1.2 Thesis outline

The following is a summary of the contents of this thesis.

1. In chapter 2, we present some background explaining our problem. We then briefly discuss some possible scenarios where our method can be applied - the low rank eigen update problem and the synthesis step in divide and conquer algorithm. In section 2.3 we summarize some of the related work in current literature.

2. In chapter 3, we give an overview of our algorithm including the overall structure and all the pieces used. We begin by introducing the bracketing algorithm and its example usage in bisection. Rayleigh Quotient Iteration and bisection are used to generate our eigenpair estimates. We discuss the convergence properties of Rayleigh Quotient Iteration and the situations where bisection is used as a backup strategy. In the last section of this chapter we mention some numerical difficulties for computing orthogonal eigenvectors.

3. In chapter 4, we show in detail some of the factorization techniques used to efficiently solve the shifted linear system in Rayleigh Quotient Iteration. The materials of this chapter represent the major advance towards our $O(n^2)$ algorithm. Due to the low rank structure of our problem, a deflation strategy is used to pre-process the input matrices and compute some eigenpairs if certain conditions are met. chapter 4.2 gives the details of the LDL^T decomposition we use to solve the linear system. Although it suffers from the same numerical instability as LU, it is used only for efficiency. We also provide a QR-based method to ensure stability of our solution to the shifted linear system.

4. In chapter 5, we discuss the QR-based fall back decomposition method for solving the shifted linear system in Rayleigh Quotient Iteration. We begin by introducing some backgrounds of QR decomposition and analysis on current QR algorithms. Then we introduce our choice of decomposition algorithm, which is based on Householder reflections. A common challenge for eigensolvers is computing orthogonal eigenvectors corresponding to eigenvalue clusters. In order to conquer this challenge, we present an *orthogonal deflation* method that avoids using any kind of re-orthogonalization method as used in conventional software packages like LAPACK [2]. We finish this chapter by giving details and complexity analysis of the proposed methods.

5. In chapter 6 we discuss how the accuracy requirements (1.1) and (1.2) are achieved for all eigenpairs. Since our algorithm uses bisection and inverse iteration, we also discuss the convergence, stopping criteria and the effect of finite precision arithmetic on the eigenvectors computed by inverse iteration. Extended precision with a double double arithmetic implementation is used to ensure orthogonality of eigenvectors when eigenvalue clusters have small absolute gaps. In the last section of this chapter we give the backward stability analysis of computed eigensystem based on equations (1.1) and (1.2).

6. In chapter 7 we present some numerical results of our algorithm. We show that the accuracy and complexity are as demanded. In addition, we include the average number of Rayleigh Quotient Iteration needed for each eigenpair as well as the total number of *QR-based* decomposition used. We finish this chapter with some concluding remarks and our thoughts on future development.

1.3 Notation

In this section we aim to clarify notations used throughout the thesis. The reader might benefit from reviewing this section occasionally throughout the reading of this thesis. We adopt the convention of denoting matrices with upper roman letters such as A, B, D, U, H and scalars by lowercase greet letters such as $\alpha, \lambda, \eta, \gamma$ or lowercase roman letters as a, b. In particular, A and H will denote symmetric matrices and D a diagonal matrix while L denotes a lower triangular matrix. Overbars and hats will be frequently used when transformations of matrices are being considered, *e.g.* $\hat{A}$ and $\hat{H}$. For slicing matrices, we adopt the convention used by MATLAB so that the submatrix of A in rows i through j will be denoted as $A(i:j,:)$ whereas the ith column of A is denoted by $A(:,i)$. Entries of matrices are denoted with double index in parentheses, *i.e.* $A(i,j)$ represents the ith row, jth column of A. Double subscripts are used to denote entries in blocked matrices. For example, we use A_{22} denote the 2nd row, 2nd column block of a blocked matrix.

Vectors are denoted by lowercase roman letters such as u, v, h. The ith component of a vector v will be denoted by $v(i)$ and the slice of components through i and j is denoted by $v(i : j)$. Unless otherwise specified, we use n to denote the size of symmetric matrices and length of vectors. For the case where a matrix has only n non-trivial entries, we will use only one index to denote those entries. Specifically, we use $D(i)$ to denote the ith diagonal entry of D, *i.e.* $D(i, i)$.

The n eigenvalues are denoted by $\lambda_1, \lambda_2, \ldots, \lambda_n$ and their corresponding eigenvectors are $q_1, q_2, \ldots, q_n$. Q and Λ will be used to denote the matrix with columns as eigenvectors and the diagonal matrix with eigenvalues on the diagonal. The computed values are denoted by adding a hat, *i.e.* $\hat{\lambda_1}$ is the computed value of λ_1.

For discussion of performance, ϵ is used to denote machine epsilon. However, it is not limited to IEEE's single or double precision arithmetic as we use it to denote a generic precision. In error analysis, it will also be used to represent "the order of machine epsilon", *i.e.* $O(\epsilon)$ to be our desired accuracy. Similarly, we will continue to "abuse" the "big Oh" notation in our computational complexity discussions. Normally, the O notation implies a limiting process. For example, when we say an algorithm is $O(n^2)$, the meaning is that it performs less than Kn^2 operations when $n \to \infty$. To be more precise, $f(n)$ is $O(n^2)$ if there exist n_0, K such that $f(n) \leq Kn^2, \forall n > n_0$. However in most of our discussions, we do not imply a limiting process but instead use it as a synonym of "order of magnitude". Our usage should be clear from the context.

At last we would like to point out some sloppy usage of the terms "eigenvalues" and "eigenvectors". For example, we use phrases such as "the computed eigenvalues are close to the exact eigenvalues" or "the computed eigenvectors are numerically orthogonal". Or we may use "approximated eigenvalues and eigenvectors". In both of the above phrases, "computed/approximated" values refer to the eigenvalues or eigenvectors computed by an algorithm. And we some times use "orthogonal" to mean "numerically orthogonal", *i.e.* "orthogonal with respect to machine epsilon". Again, such usage should be clear based on the context it's in.

Chapter 2

Problem background and motivations

In this chapter, we start by giving a quick background to the problem of computing eigenvalues and eigenvectors of symmetric matrices. We then discuss the most commonly used tridiagonalization technique and how the increase in relative cost of tridiagonalization makes symmetric band matrix an interesting topic. Later in chapter 2.2 we show the two main motivations to our proposed method, including how our method fits in the symmetric band matrix eigenproblem and how it solves the low rank modification problem.

2.1 Background

Computations of eigensystems are seen in a variety of contexts, ranging from chemistry to economics. Quantum chemists need to compute eigenvalues to reveal the electronic energy states in a large molecule [20], an engineer may need to solve problems concerning natural frequencies of objects. In addition, eigenvalues can convey information about the market in some models for economists. Lots of such meaningful realistic problems can be abstracted to a mathematical problem of finding all numbers λ and non-zero vectors q such that:

$$Aq = \lambda q,$$

where A is a real symmetric matrix of order n, λ is called an *eigenvalue* of A and q is called the corresponding *eigenvector*.

By definition, all eigenvalues of A must satisfy the characteristic equation $det(A - \lambda I) = 0$, which is a polynomial of order n and hence always has n roots in the complex domain. A symmetric matrix further enjoys the following two properties:

1. All eigenvalues are real;

2. There exists n orthogonal eigenvectors.

Hence, a symmetric matrix A has an *eigendecomposition*:

$$A = Q\Lambda Q^T,$$

where $\Lambda = diag(\lambda_1, \lambda_2, \ldots, \lambda_n)$ is a diagonal matrix containing all the eigenvalues and Q is an orthogonal matrix of eigenvectors, i.e., $Q^TQ = I$.

There is a long history of methods for such *eigendecompositions*. Explicitly forming and solving the characteristic equation seems to have been a popular early choice. However, roots of such polynomials are very unstable and extremely sensitive to changes in the coefficients and the inadequacy of finite precision arithmetic in modern computers further enlarged this problem. Orthogonal matrices and orthogonal transformations, with its stability and property of preserving eigenvalues became and still remains a popular choice. i.e., a sequence:

$$A_0 = A, \quad A_{i+1} = Q_i^T A_i Q_i,$$

where $Q_i^TQ_i = I$. After realizing the impossibility to transform A directly into a diagonal matrix, it seemed a natural attempt to transform A into a tridiagonal matrix instead. Givens first proposed a method for such transformation using plane rotations [24]. However, most modern algorithms reduce A to a tridiagonal matrix T by a sequence of $n-2$ reflections, now named after its inventor Householder [30]. Mathematically:

$$T = (Q_{n-2}^T \ldots Q_2^T Q_1^T) A (Q_1 Q_2 \ldots Q_{n-2}) = Z^T A Z.$$

The *eigendecomposition* of T may now be found as:

$$T = V\Lambda V^T,$$

where $V^TV = I$. Since orthogonal transformations preserves eigenvalues, Λ already contains all eigenvalues of A. A process called *back-transformation* is used to find the eigenvectors:

$$A = (ZV)\Lambda(ZV)^T = Q\Lambda Q^T.$$

Tridiagonal eigen-decomposition problem is one of the most heavily researched topic in numerical linear algebra. A variety of algorithms exploit the tridiagonal structure and remain in the core part of state-of-the-art software packages such as EISPACK [53] and LAPACK [2]. However, the advancement of modern computer architecture and the increase in relative costs to move data between memory layers have post new concerns for tridiagonalization. In most situations, tridiagonalization can be relatively more expensive in comparison to the costs of computing the eigenpairs. For general real symmetric matrices, tridiagonalization and reconstruction process can take up 90% of the running time if only eigenvalues are calculated and 50% if both eigenvalues and

eigenvectors are required [6]. Moreover, unfavorable data access patterns and bad data locality can cause even more inefficiencies in the tridiagonalization process. The above effects are only even worse for symmetric band matrix because its tridiagonalization process is completed by the annihilate-and-chase method, which does not make use of BLAS-3 operations [6]. This fact has motivated the idea of a "*two-step*" reduction - reducing the symmetric matrix A to a symmetric band matrix B, then reducing B to a tridiagonal matrix T has become popular. Extensive research has gone into this area and several successful attempts have been made [35].

Given the importance of eigenproblem for symmetric band matrices, researchers have devoted extensive efforts into it. Currently, the standard method to compute all eigenpairs of a symmetric band matrix, for example as implemented in LAPACK [2], is to tridiagonalize the band matrix via orthogonal transformations, then solve the similar tridiagonal eigenproblem and use back-transformation to get the original eigenpairs. The standard *divide-and-conquer* algorithm for finding all eigenpairs of a tridiagonal matrix, which was first proposed by Cuppen [16], has made such approaches for symmetric band eigenproblem more favorable. The core of this algorithm is a method to efficiently find all eigenvalues and eigenvectors of a diagonal matrix with a rank-one modification. In addition, a phenomenon called *deflation* was exploited to boost the performance of the algorithm. Efficient and numerically stable implementations of Cuppen's algorithm have been developed overtime.

Although it exhibits obvious favorable properties in parallelization, the divide and conquer approach combined with tridiagonalization is even sequentially one of the most efficient algorithms for finding the eigensystem of a large dense symmetric band matrix. [18, Chapter 5]

Inspired by the tridiagonal case, attempts have been made to apply this divide and conquer idea directly to symmetric band matrices. In fact, it can be generalized to another form called block tridiagonal matrix:

$$B = \begin{bmatrix} B_1 & C_1^T & & & \\ C_1 & B_2 & C_2^T & & \\ & C_2 & B_3 & \ddots & \\ & & \ddots & \ddots & C_{p-1}^T \\ & & & C_{p-1} & B_p \end{bmatrix}$$

where the diagonal blocks B_i are symmetric and the off-diagonal blocs C_i are arbitrary. Symmetric band matrices can be represented by restricting C_is to upper-triangular matrices. Various attempts to port over the divide and conquer algorithm have been made through this generalization, from Golub, Arbenz and Gander [45, 5, 4] to more recently by Gansterer et.al [23]. Many theoretical aspects have been investigated and eigenvalue computations can be done accurately and stably. However, the main problem remains in numerical stability of computed eigenvectors.

Another attempt of divide and conquer method has succeeded for a special case of block tridiagonal matrices, *i.e.* the case where C_is are rank-one matrices. The result presented in [22] shows that the algorithm is highly efficient and numerically stable. In general, the off-diagonal matrices C_i are not rank-one for block tridiagonal matrices arising from applications. Approximating the off-diagonal matrices with rank-one matrices is often not sufficiently accurate. Gansterer et.al [23] have also proposed to compute approximate eigenpairs of symmetric band matrices and have achieved favorable results. A detailed comparison of methods for eigenproblem of symmetric band matrices is presented in [37].

2.2 Motivations

In various attempts to generalize the divide and conquer approach to symmetric band matrices, the central problem has appeared to be computing the eigensystem of a diagonal matrix with symmetric low-rank modification, similar to the rank-one modification in tridiagonal case. The algorithm we propose in this thesis is aimed to solve this problem. In this section we list two important motivations for our algorithm.

2.2.1 Rank r modification of symmetric eigensystem

Golub [45] first proposed some theoretical concerns for the problem of finding eigenvalues and vectors of the matrix $A + VV^T$ where A is a symmetric matrix with known spectral decomposition and VV^T is a positive semi-definite matrix of low rank. This thesis instead looks at a generalized version of the problem without requiring the modification to be positive semi-definite.

Formally, let

$$A = QDQ^T$$

where $Q = [q_1, q_2, \ldots, q_n]$ is an orthogonal matrix and $D = diag(\lambda_1, \lambda_2, \ldots, \lambda_n)$ is diagonal. Here $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$ are the eigenvalues and $q_1, q_2, \ldots q_n$ are the corresponding eigenvectors. Let $V \in \mathbb{R}^{n\times r}$ be a column orthogonal matrix and $H \in \mathbb{R}^{r\times r}$ be a rank-r symmetric matrix. In this thesis we consider the problem to compute the eigenvalues $\tilde{\lambda_1}, \tilde{\lambda_2}, \ldots, \tilde{\lambda_n}$ and the corresponding eigenvectors $\tilde{q_1}, \tilde{q_2}, \ldots, \tilde{q_n}$ of the *modified eigenproblem*:

$$\tilde{A}x := (A + VHV^T)x = \tilde{\lambda}x.$$

Or equivalently:

$$(D + UHU^T)y = \tilde{\lambda}y, \quad V = QU, \quad x = Qy.$$

One example of such problem occurs if the eigenvalues of an operator such as Laplacian or biharmonic operator are to be determined by difference methods on a domain that's decomposible into small sub-domains on which discretized eigenvalue

problem can be solved easily [45, 5]. Typically, the matrix from difference methods have the following block structure:

$$A = \begin{bmatrix} T_1 & 0 & \dots & 0 & R_1 \\ 0 & T_2 & 0 & 0 & R_2 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & & T_k & R_k \\ R_1 & R_2 & \dots & R_k & Q \end{bmatrix}$$

where T_i represents the difference operator on the sub-domains. R_i and Q, caused by the boundary conditions and interface points, usually have relatively small dimension r compared to the dimension n of matrix A [45]. Since the eigenvalues of T_i are easily computed, it is natural to split A as $diag(T_1, T_2, \dots, T_k, Q)$ plus a rank $2r$ modification.

2.2.2 Divide and conquer algorithm for symmetric band matrix

Another important problem of this type is the rank one modification of the tridiagonal problem as proposed by Cuppen [16]. However, the same idea cannot be readily applied to general symmetric band matrix because the eigenproblem of multiple rank modification of a diagonal matrix cannot be solved with similar ideas.

Consider an $n \times n$ symmetric matrix with semi-bandwidth r that is divided into $n/p \times n/p$ blocks according to:

$$B = \begin{bmatrix} B_1 & R_1^T & & & & \\ R_1 & B_2 & R_2^T & & 0 & \\ & R_2 & B_3 & R_3^T & & \\ & & \ddots & \ddots & \ddots & \\ & 0 & & R_{p-2} & B_{p-2} & R_{p-1}^T \\ & & & & R_{p-1} & B_p \end{bmatrix}$$

The diagonal blocks $B_i \in \mathbb{R}^{n/p \times n/p}, i = 1, 2, \dots, p$ are symmetric and banded with bandwidth r. The subdiagonal blocks $R_i \in \mathbb{R}^{r \times r}$ are upper triangular and non-singular.

For simplicity, the case $p = 2$ will be examined in detail and general case follows easily. Let

$$B = \begin{bmatrix} B_1 & H_1^T \\ H_2 & B_2 \end{bmatrix} = \begin{bmatrix} B_1 & \\ & B_2 \end{bmatrix} + R, \quad B_i \in \mathbb{R}^{n/p \times n/p}$$

Arbenz [5] shows in his work that R can be represented by a special form:

$$R = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & M & R_1^T & 0 \\ 0 & R_1 & R_1 M^{-1} R_1^T & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

where $M \in \mathbb{R}^{r\times r}$ can be chosen differently. One choice of M used in Arbenz's work [5] is based on the singular value decomposition (SVD) of R_1. Let $R_1 = U\Sigma V^T$, then $M = -V\Sigma V^T$ and $R_1 M^{-1} R_1^T = -U\Sigma U^T$. So both matrices are symmetric negative definite and have the same norm and condition number as R_1. Then the above expression becomes:

$$B = \begin{bmatrix} B_1 & \\ & B_2 \end{bmatrix} + ZHZ^T,$$

where $Z = \begin{bmatrix} 0 \\ U \\ V \\ 0 \end{bmatrix}, \quad H = \Sigma.$

Let $B_i = Q_i D_i Q_i^T, i = 1, 2$ be the eigendecompositions of the diagonal blocks, the synthesis problem to combine these two solutions becomes:

$$B = \begin{bmatrix} Q_1 & \\ & Q_2 \end{bmatrix} \left(\begin{bmatrix} D_1 & \\ & D_2 \end{bmatrix} + UHU^T \right) \begin{bmatrix} Q_1 & \\ & Q_2 \end{bmatrix}^T,$$

where $U = \begin{bmatrix} Q_1 & \\ & Q_2 \end{bmatrix}^T Z.$

Letting $D = \begin{bmatrix} D_1 & \\ & D_2 \end{bmatrix}$ makes the synthesis problem the eigenproblem of the matrix $A = D + UHU^T$, with H being a diagonal matrix itself. The problem that this thesis approaches is then a generalization of low-rank modification of diagonal matrix for a general symmetric non-singular H. An efficient algorithm for this problem can potentially lead to an efficient and stable divide and conquer algorithm for the eigenproblem of band symmetric matrices.

2.3 Related work

Arbenz, Gander and Golub [45] first investigated into computing the eigensystem of symmetric band matrices. The authors provide many important results about the analysis of low-rank modifications of a diagonal matrix. Two methods were proposed by Arbenz [4] for computing the eigensystem of a diagonal matrix with symmetric rank-r modification. The first method approaches the rank-r modification as a sequence of r rank-1 modifications, whereas the second method views the rank-r modification as another small $r \times r$ eigenproblem and uses its solution to solve the original rank-r modification problem. If both eigenvalues and eigenvectors are required, the first approach has high algorithmic complexity ($O(n^3)$), but the second method was found to suffer from numerical stability issues for eigenvector computations.

Various other attempts have been made regarding the divide and conquer algorithm for symmetric band matrices. Gansterer et.al [59] proposed a low-complexity method

for the eigensystem of symmetric band matrix. But they proposed to compute the eigenvectors separately using a modified QR iteration aside from the eigenvalue computation. HaiDar et.al [27] also proposed a divide and conquer algorithm for symmetric matrices which relies on an eigensolver for symmetric band matrix. The synthesis problem to find the eigensystem of a rank-r modified diagonal matrix was approached by r rank-one modifications in their proposed implementations and thus, their method is only viable for eigenvalue computations.

Rank-2 modification of a diagonal matrix has been analyzed by HyungSeon Oh and Zhe Hu [39], and a similar approach is suspected for general rank-r modification. The special case for divide an conquer algorithm on block tridiagonal matrix with rank-one off-diagonal blocks has been proposed with a numerically stable implementation. However, most symmetric band matrices aroused in application does not have rank-one off-diagonals and approximating them with rank-one matrices are not always sufficiently accurate. A method for approximating the eigenpairs of such rank-r modification has been studied by Gansterer et.al [59] as well.

The algorithm proposed in this thesis is motivated by the difficulty in computing eigenvectors for rank-r modified diagonal matrices as well as its application in divide and conquer algorithm for symmetric band eigenproblem. Our algorithm tries to utilize the special structure of the modification for fast Rayleigh Quotient Iteration to find both the eigenvalues and eigenvectors at the same time. The stability issue of eigenvectors are approached by a combination of *Orthogonal Deflation* technique and extended precision arithmetic when necessary.

Chapter 3

Algorithm Overview

The algorithm proposed in this thesis takes root from a broader kind of algorithm called "bracketing". The main idea is to start from an interval containing all the eigenvalues of $A = D+UHU^T$, then generate a sequence μ_k of eigenvalue estimates that will divide the interval into subintervals until they are too narrow. To improve overall efficiency, our algorithm also generates a sequence x_k of eigenvector estimates such that (μ_k, x_k) converges to an eigenpair while dividing the intervals.

The eigenpair sequence (μ_i, x_i) is generated by Rayleigh Quotient Iteration:

$$\begin{cases} x_{k+1} = & (A - \mu_k I)^{-1} x_k \\ x_{k+1} = & \frac{x_{k+1}}{||x_{k+1}||_2} \\ \mu_{k+1} = & x_{k+1}^T A x_{k+1} \end{cases}$$

In chapter 3.1 we introduce general bracketing algorithm as well as our choice of iteration method that generates a convergent sequence to an exact eigenpair. The strategy used to divide intervals while generating the sequence is also included. Given our choice of Rayleigh Quotient Iteration, we discuss the methods used to efficiently and stably solve the shifted linear system involved. Beginning by setting the objectives of our solution methods, we proceed to examine some existing techniques and their drawbacks for our use case. and conclude by introducing our choice of decomposition methods used to solve the shifted linear system. In chapter 3.3, we briefly talk about potential numerical difficulties in the accuracy of computed eigenpairs and orthogonality of eigenvectors.A more thorough discussion is presented in chapter 6. We finish this chapter by presenting a description of high-level implementation of our bracketing algorithm in 3.4.

3.1 Bracketing

Bracketing is an algorithm that depends on a function $Count(x)$ to count the number of eigenvalues of A smaller than x. Then it is easy to see that the number of eigen-

values in a half open interval $[\alpha_1, \alpha_2)$ is equal to $Count(\alpha_2) - Count(\alpha_1)$. A general bracketing algorithm can refer to any one that involves dividing an interval containing at least one eigenvalue into subintervals of any size and recomputing the number of eigenvalues in each subinterval. The algorithm terminates when the subintervals are narrow enough [19].

In order to compute all the eigenvalues and eigenvectors, bracketing algorithm is usually combined with another eigenvector solver such as inverse iteration. Bisection, which is proposed by Wallace Givens in 1954 [24] is a good example for bracketing algorithm. It permits an eigenvalue to be computed in $O(n)$ flops and thus takes $O(n^2)$ flops to compute all eigenvalues. Faster iteration methods that exhibit superlinear convergence properties also exist such as Laguerre's method [56, 42] and Zeroin scheme [48].

Once an approximated eigenvalue $\hat{\lambda}$ is computed, the method of inverse iteration may be used to compute the corresponding eigenvector approximation:

$$x_0 = b, \quad (A - \hat{\lambda}I)x_{k+1} = \tau_k x_k, \quad k = 0, 1, 2, \ldots,$$

where b is the starting vector, x_k denotes the eigenvector estimate and τ_k is the normalization scalar for the kth iteration. There were early fears about the loss of accuracy in solving the above linear system due to $A - \hat{\lambda}I$ being near-singular when $\hat{\lambda}$ is an accurate approximation to an exact eigenvalue, but it was later showed that this lack of accuracy would not affect the computed eigenvectors [46]. We will present some analysis relevant to our problem in chapter 3.3 and 6. Overall, inverse iteration delivers an approximated eigenvector $\hat{u}$ with a small residual whenever $\hat{\lambda}$ is close to the exact eigenvalue λ, *i.e.* small norm $||(A - \hat{\lambda})\hat{u}||$. Although the small residual implies a small backward error for computed eigenvectors, it does not guarantee orthogonality when eigenvalues are close together. For symmetric matrices, when several computed eigenvalues $\hat{\lambda}_1, \hat{\lambda}_2, \ldots, \hat{\lambda}_k$ are close together, their corresponding computed eigenvectors $\hat{q}_1, \hat{q}_2, \ldots, \hat{q}_k$ may not be orthogonal. In this case, *re-orthogonalization* is needed by computing the QR decomposition of $[\hat{q}_1, \hat{q}_2, \ldots, \hat{q}_k] = QR$ and replacing each $\hat{q}_j$ with the jth column of Q; this guarantees the orthogonality of corresponding eigenvectors.

For an improvement in overall efficiency of bracketing, we seek to generate a sequence of eigenpair estimates that converges to a real eigenpair while dividing the intervals. We pick Rayleigh Quotient Iteration for its fast convergence.The properties of Rayleigh Quotient Iteration are well-studied and summarized in the following theorem:

Theorem 3.1 (Global convergence) *[43] Let (μ_k, x_k) be the sequence generated by RQI and $r_k = (A - \mu_k I)x_k$ be the kth residual.*

The sequences $\{||r_k||\}$ and $\{\mu_k\}$ always converge. However, there are two possible cases to consider. One possibility is that

$$\lim_{k \to \infty} ||r_k|| = 0.$$

In this case the sequence μ_k converges toward an eigenvalue λ_j of A, and $\{x_k\}$ converges in the direction of the corresponding eigenvector. The second possibility is that

$$\lim_{k\to\infty} ||r_k|| = \eta > 0 \tag{3.1}$$

This case is characterized by the following features:

- *(a) The sequence μ_k converges toward a point $\mu^* = (\lambda_i + \lambda_j)/2$, where λ_i and λ_j are two distinct eigenvalues of A;*
- *(b) $\lim_{k\to\infty} x_{2k} = x^*$ and $\lim_{k\to\infty} x_{2k+1} = x^{**}$ where $x^* \neq x^{**}$;*
- *(c) The limit vectors x^* and x^{**} are eigenvectors of $(A - \mu^* I)^2$ but not of A;*
- *(d) The second situation is "unstable" under small perturbation of x_k.*

The "instability" in the second case means that as $||r_k||$ approaches η, small perturbations in x_k can possibly reduce $||r_k||$ under η. Once $||r_k||$ becomes smaller than η, the decreasing property of $||r_k||$ drives it down to zero [44]. This "instability" also introduces unpredictability of the behavior of Rayleigh Quotient Iteration since the sequence can potentially converge to different limit points due to small perturbations. This unpredictable behaviour is perhaps the reason why Rayleigh Quotient Iteration has never been considered a practical way for computing a complete eigensystem. In our proposed algorithm, bisection is used as a backup strategy when Rayleigh Quotient Iteration does not converge to an eigenpair or converges to a known eigenpair.

In order to divide the intervals in our *Worklist*, whenever a new eigenvalue estimate μ_k is generated, by Rayleigh Quotient Iteration or by bisection, it is located in one of the intervals in *Worklist* with *Count(μ_k) i.e.* the number of eigenvalues smaller than μ_k computed. Then the interval $[\alpha, \beta)$ containing μ_k is divided into $[\alpha, \mu_k)$ and $[\mu_k, \beta)$. The number of eigenvalues in both intervals are calculated as $Count(\mu_k) - Count(\alpha)$ and $Count(\beta) - Count(\mu_k)$ respectively.

Rayleigh Quotient Iteration will be stopped when an estimate (μ, x) generates a small residual $||(A - \mu)x||$. The following theorem justifies the stopping criteria in the sense of backward error:

Theorem 3.2 (Backward error bound) *Let u be any nonzero vector in $\mathbb{R}^n$ such that $||u||_2 = 1$. Let $\rho = \rho(u) = u^T Au$ denote the corresponding Rayleigh Quotient and let*

$$r = Au - \rho u,$$

denote the corresponding residual. Then there exists an eigenvalue λ of A satisfying:

$$|\lambda - \rho| \leq ||r||_2.$$

Theorem 3.2 and the local cubic convergence property suggest that Rayleigh Quotient Iteration be terminated as soon as $||r||_2$ falls below a certain threshold value. Our choice of the threshold value and a further use of $||r||_2$ to bound the the deviation of eigenvectors from orthogonality will be discussed in Chapter 6.

3.2 Decomposition methods

Traditionally, one of the main difficulties for making Rayleigh Quotient Iteration efficient is solving the shifted linear system as the system changes every iteration. When used to generate sequence of estimates in a bracketing algorithm, it is required that the iteration also outputs *Count* of the shift, namely the number of eigenvalues smaller than the shift. The following summarizes the objectives of our decomposition method used to solve the shifted linear system:

1. Efficiency. Given that Rayleigh Quotient Iteration takes constant number of iterations to converge for a single eigenpair, it takes $O(n)$ number of iterations for the whole eigensystem. Thus, each iteration needs to at least achieve a better asymptotic complexity than $O(n^2)$ to make the overall complexity feasible.

2. Stability. The solution of the shifted linear system $(A - \hat{\lambda}I)\hat{x} = b$ needs to be backward stable in every iteration. Namely, we would like to achieve a small backward error:
$$\frac{||(A - \hat{\lambda}I)\hat{x} - b||}{||A||||\hat{x}||}.$$

3. Computation of *Count*($\hat{\lambda}$). Our decomposition method needs to also compute the number of eigenvalues of A that are smaller than $\hat{\lambda}$ to update the number of eigenvalues in each interval after we divide the interval in the *Worklist* by $\hat{\lambda}$.

Gaussian elimination with partial pivoting in conjunction with back substitution can be used to obtain solution of a linear system $Ax = b$ for any non-singular A. Plain Gaussian elimination assumes a general matrix and takes $O(n^3)$ to complete, which is out of the scope of our efficiency requirement. For a banded matrix, Gaussian elimination can be adapted to be much more efficient. For example,a symmetric band matrix A with constant bandwidth b can be decomposed in $O(n)$ time [34] provided that no pivoting is used. In this case the factors L and U in the LU decomposition $A = LU$ are both banded matrices with bandwidth b and the subsequent back substitutions can also be completed in $O(n)$ time.

Variants of Gaussian elimination can have better efficiency when applied to symmetric matrices. The Cholesky decomposition $A = LL^T$ [15], for example, can cut the number of flops by half for a symmetric positive definite matrix A. However, the

positive definiteness of the shifted matrices $A - \hat{\lambda}$ depends on the shift $\hat{\lambda}$ and thus is not guaranteed in our use case.

Our decomposition method is based on another variation of Gaussian elimination on symmetric matrices called LDL^T decomposition [34], or "square-root-free Cholesky factorization". Similar to Gaussian elimination, it can be modified to gain efficiency when applied to a banded matrix. More importantly, the matrix D has the same inertia with $A - \hat{\lambda}$ and readily gives $Count(\hat{\lambda})$ as the number of negative diagonal entries. We present a more detailed discussion of this variation in chapter 4.2.

When stability is concerned, Gaussian elimination and its variants adopt pivoting strategies, which require comparison of elements in columns and sometimes rows. In our case, the entries in $A - \hat{\lambda}I = D - \hat{\lambda}I + UHU^T$ are not directly given and thus pivoting requires computation of the entries beforehand. Since this pre-computation adds significant overhead to the decomposition method and numerical instability happens very rarely, we adopt the strategy to keep the method efficient but unstable for most of the time and fallback to a more stable solution when needed. In the case of a solution with large backward error, we will recompute it with a QR-based decomposition algorithm to guarantee numerical stability. The method is described in detail in chapter 5.

In summary, our decomposition method is a combination of an efficient but unstable LDL factorization with a slower but stable QR-based version. In both cases, the number of flops required is $O(nr^2)$ and $Count(\hat{\lambda})$ is computed. Although the stable version has a larger constant hidden in the "big oh" notation, it is only used very occasionally.

3.3 Numerical difficulties

The quality of an approximate eigenvector x is measured by its residual with the following result [51]:

Theorem 3.3 *Let A be a real symmetric matrix with λ being a simple eigenvalue. Let v be a normalized eigenvector, then for any vector x and scalar μ, closer to λ than any other eigenvalue:*

$$|\sin \angle(v, x)| \leq ||Ax - \mu x||/gap(\mu),$$

where $gap(\mu) = \min |\gamma - \mu| : \gamma \neq \lambda, \gamma \in spectrum(A)$. In addition, the error in the eigenvalue is also bounded by the residual norm, i.e:

$$|\mu - \lambda| \leq ||Ax - \mu x||.$$

The orthogonality of computed eigenvectors also depends on the residual norm. Suppose v_1, v_2 are exact eigenvectors with approximations x_1, x_2, if:

$$|\sin \angle(x_1, v_1)| \leq n\epsilon||A||, \quad |\sin \angle(x_2, v_2)| \leq n\epsilon||A||$$

where ϵ is a given machine precision, then

$$|\cos\angle(x_1, x_2)| \leq |\sin\angle(v_1, x_1)| + |\sin\angle(v_2, x_2)| \leq 2n\epsilon||A||.$$

In general, the best we can hope for is to produce a residual $r = Ax - \mu x$ whose norm is relatively comparable to machine precision, *i.e.*:

$$||r|| \leq n\epsilon||A||.$$

By the above results, , if $gap(\mu) > tol * ||A||$, where tol is a gap threshold (say 10^{-3}), then

$$|\sin\angle(v, x)| \leq n\epsilon/tol.$$

and accuracy is ensured. Many eigensolvers (*i.e.* inverse iteration) can produce accurate and numerically stable eigenvectors if the eigenvalues are separated by a large enough absolute gap, i.e $gap > tol * ||A||$. On the other hand, in the case where $gap < tol * ||A||$, the residual norm must be much smaller than $n\epsilon||A||$ in order to deliver the same accuracy.

In our proposed algorithm, extended precision arithmetic is employed in some parts of the computation in the case of a small absolute gap. The implementation is done with double double arithmetic to simulate quadruple precision for efficiency. Detailed analysis of accuracy and orthogonality is presented in chapter 6.

3.4 An implementation of the bracketing algorithm

Our algorithm is similar to other bracketing algorithms in that we maintain a worklist of tasks and find eigenvalues one at a time while splitting the intervals. We define *task* to be a 4-tuple $T = (\alpha, \beta, n_\alpha, n_\beta)$, where $[\alpha, \beta]$ is a non-empty interval, n_α and n_β are the *Counts* associated with α and β respectively, i.e the number of eigenvalues of A smaller than α, β.

Unlike usual bracketing algorithm, the aforementioned unpredictable behaviour of Rayleigh Quotient Iteration means that the estimates could converge or land anywhere, even outside of all intervals in our *Worklist*. Hence, bisection is used as a backup strategy to ensure that progress is made in every iteration, *i.e.* at least an interval gets a split.

Another challenge for Rayleigh Quotient Iteration is efficiently and stably solving a shifted linear system in every iteration. The *RQI* subroutine handles this with a decomposition method and the details are discussed in chapter 4 and 5.

In our implementation, a new estimate of eigenpair is first generated every iteration by Rayleigh Quotient Iteration, i.e the *RQI* subroutine in the pseudo-code:

$$\begin{cases} x_{k+1} = & (A - \mu_k I)^{-1} x_k \\ x_{k+1} = & \frac{x_{k+1}}{||x_{k+1}||_2} \\ \mu_{k+1} = & x_{k+1}^T A x_{k+1} \end{cases}$$

Algorithm 3.4.1 The algorithm to find all eigenpairs

1: **procedure** BRACKET($n, r, D, U, H, left, right, n_{left}, n_{right}, \tau, W, Q$)
2: ▷ Computes all eigenvalues of $D + UHU^T$ in the interval $[left, right]$ and their corresponding eigenvectors to the desired accuracy τ. The initial task $(left, right, n_{left}, n_{right})$ is given.
3: **if** ($n_{left} \geq n_{right}$ .or. $left > right$) **then**
4: return
5: **end if**
6: enqueue $(left, right, n_{left}, n_{right})$ to *Worklist*
7: DEFLATE($D, U, deflate, W, Q$) ▷ Number of deflated eigenpairs = $deflate$
8: **while** *Worklist* is not empty **do**
9: Pick $(\alpha, \beta, n_\alpha, n_\beta)$ from the *Worklist*
10: μ, x = INITIALGUESS($n, r, D, U, H, \alpha, \beta$)
11: multiplicity = 0
12: **while** not converged **do**
13: x', μ', n_μ = RQI(D, U, H, μ, x)
14: residual = $|(D + UHU^T - \mu' I)x||$
15: **if** μ is not found in any interval **then**
16: BISECTION(n, r, D, U, H, multiplicity)
17: **if** (multiplicity > 0) **then**
18: exit
19: **end if**
20: **else**
21: Dequeue $(\alpha, \beta, n_\alpha, n_\beta)$ from *Worklist* such that $\mu \in [\alpha, \beta]$
22: **end if**
23: **if** residual $< \tau$ .and. μ' is found in some interval **then**
24: HANDLECONVERGENCE(*Worklist*, n, r, D, U, H, μ, x, multiplicity)
25: exit
26: **end if**
27: **if** ($n_\mu > n_\alpha$) **then**
28: enqueue $(\alpha, \mu, n_\alpha, n_\mu)$ to *Worklist*
29: **end if**
30: **if** ($n_\mu < n_\beta$) **then**
31: enqueue $(\mu, \beta, n_\mu, n_\beta)$ to *Worklist*
32: **end if**
33: $\mu = \mu'$, $x = x'$
34: **end while**
35: **if** multiplicity > 1 **then**
36: HANDLECLUSTER($n, r, D, U, H, \mu, x, W, Q$, multiplicity)
37: **else**

```
38:             W(deflate + 1) = μ
39:             Q(defalte + 1, :) = x
40:         end if
41:         deflate = deflate + multiplicity
42:     end while
43: end procedure
```

The $Deflate$ subroutine (chapter 4.1) used in Algorithm 3.4.1 is a pre-process method to detect and compute certain eigenvalues and vectors if the input matrix satisfies certain conditions. This is similar to the one used in rank-one case [18]. $InitialGuess$ is a subroutine to pick a random unit vector with its Rayleigh Quotient as the starting eigenpair estimate. $HandleConvergence$ subroutine(chapter 5.4) checks the multiplicity of converged eigenpair and enqueues the remaining tasks to *Worklist*. $HandleCluster$ is a subroutine that finds eigenvectors corresponding to an eigenvalue cluster in a stable way, and this will be furthered discussed in section 5.3.

$Bisection(n, r, D, U, H, multiplicity)$ is the fallback strategy mentioned in section 3.1. It is used when Rayleigh Quotient Iteration converges to an eigenpair not in the *Worklist* or when RQI does not make big enough progress, *i.e.* it falls in the second situation of theorem 3.1. The method takes the first interval in the *Worklist*, uses the mid point μ' as its next eigenvalue estimate and slices the interval accordingly. As for the eigenvector estimate, it picks a random unit vector if Rayleigh Quotient Iteration already converges , or uses the solution of the shifted equation $(A - \mu' I)x = y$ as the next estimate, where y is the last eigenvector estimate. The choice is dependent on the current progress made in terms of reducing the residual $||(A - \mu I)x||$. *multiplicity* is an integer returned by *Bisection* indicating the multiplicity of the eigenvalue found in the interval, or 0 if not converged.

The computational bottleneck for our proposed algorithm is Rayleigh Quotient Iteration,i.e. the RQI method, which takes $O(nr^2)$ flops for each iteration. Given the fast convergence properties of Rayleigh Quotient Iteration, it is reasonable to assume that it converges within constant number of iterations (empirically 5 to 7). Hence, the overall complexity of our algorithm is $O(n^2r^2)$ to compute all n eigenvalues and their corresponding eigenvectors.

Chapter 4

Preliminary methods

In this chapter, we discuss the details of some basic and efficient methods used in our algorithm. We begin in chapter 4.1 by introducing a deflation pre-process technique used to simplify the eigenproblem when certain conditions are met. This is similar to the deflation method used in the divide and conquer algorithm for tridiagonal eigenproblem. In addition to boosting performance, this pre-process procedure ensures that the input after deflation has an upper bound on possible eigenvalue cluster size, *i.e.* multiplicity of eigenvalues. For symmetric matrices, computed eigenvectors corresponding to eigenvalues with multiplicity > 1 usually suffer from loss of orthogonality. Usually, eigensolvers use extra re-orthogonalization subroutines for such eigenvalue clusters and our algorithm is of no exception. Our algorithm adopts a deflation-based method to compute numerically orthogonal eigenvectors for such eigenvalue clusters. The upper bound on eigenvalue cluster size ensures that such special method is not extensively used and the overall efficiency is not affected much. The details of this method can be found in chapter 5.3.

Later, we introduce the decomposition method used to solve the shifted linear system in Rayleigh Quotient Iteration and compute *Count(*$\hat{\lambda}$*)* for bracketing algorithm 3.4.1. Our objectives and concerns about the choice of decomposition method have been mentioned in chapter 3.2. We pick a modified version of LDL^T decomposition as it fulfills our efficiency requirement, computes *Count(*$\hat{\lambda}$*)* with no extra effort and is stable most of the time. In chapter 4.2 we explain how the input matrix could be decomposed recursively one column at a time and thus achieve a favorable data re-use pattern.

Similar to LU decomposition, it suffers from numerical instability. More over, since our shifted matrix $A-\hat{\lambda}I$ is indefinite, the diagonal entries of D in LDL^T decomposition is not necessarily positive. We use a small threshold value to replace the diagonal entry of D that's close or equal to zero. For numerically unstable case, the solution of the shifted linear system is recomputed using a stable QR-based decomposition method. The details can be found in chapter 5.

4.1 Deflation pre-process

Deflation was used in the tridiagonal Divide and Conquer eigenvalue algorithm to simplify the synthesis step and make later accumulation of eigenvectors more efficient. For rank-one case $B = D + \rho uu^T$, if two diagonal entries of D are equal, *i.e.* $D(i) = D(j)$, or when $u(i) = 0$, then $D(i)$ is an eigenvalue of B and e_i is the corresponding eigenvector [18]. One benefit being that it finds an eigenpair without computation and the other benefit is the simple form of the computed eigenvector, which makes subsequent multiplications of eigenvectors more efficient. In fact, this deflation occurs frequent enough in the divide and conquer algorithm for tridiagonal matrix such that it is one of the most efficient sequential algorithm for tridiagonal eigensystem [20].

In this section, we generalize the deflation to general low-rank case $D + UHU^T$. Similarly, it detects conditions in matrices D and U to find special eigenpairs without computation and decreases the size of the problem. In addition, the deflated input matrix has a small upper bound on its eigenvalue cluster sizes, or it can be deflated otherwise. This upper bound is crucial to the efficiency of our method for finding orthogonal eigenvectors corresponding to eigenvalue clusters.

Deflation occurs when one of the following conditions is met:

Proposition 4.1 *Let $A = D + UHU^T$,*

- *If $D(i_1) = D(i_2) = \ldots = D(i_k) = \lambda$ and $k > r$, then λ is also an eigenvalue of A with cluster size $k - r$.*
- *If $||U(i,:)|| = 0$, then $(D(i), e_i)$ is an eigenpair of A.*

Proof 4.1

- *For the first part, without lose of generality assume that $i_1 = 1, i_2 = 2, \ldots, i_k = k$. Then since U is an $n \times r$ column orthogonal matrix, the rank of U is r and the null space of $U^T(1:k,:)$ is then $k - r$.*

  If $k > r$, then the null space of $U^T(1:k,:)$ is non-empty and let $U^T(1:k,:)x = 0$. Then $D * [x, 0]^T = \lambda[x, 0]^T$ implies that $Ax = \lambda x$. The cluster size of the eigenvalue λ will be the dimension of the null space of $U^T(1:k,:)$, which is $k - r$.

- *If $||U(i,:) = 0||$, then*

$$Ae_i = d_i e_i + U^T e_i = d_i e_i.$$

  So (d_i, e_i) will be an eigenpair.

In practice, a threshold value ϵ is used to both determine if an entry in D repeats k times and to set $U(i,:) = 0$ if $||U(i,:)|| < \epsilon$. The formal deflation is completed in three steps:

1. Check the row norms of U and deflate the rows that have small norms;
2. Sort the array D, permute rows of U correspondingly;
3. Check for clusters of values in D and deflate clusters.

This deflation technique together with the low-rank structure of our matrix puts an upper bound on the eigenvalue clusters:

Proposition 4.2 (Cluster size upper bound) *Let $A = D + UHU^T$. If A has a cluster of eigenvalues (λ) with size more than $2r$, then there exist $k > 0$ and indices $i_1, i_2, \ldots, i_{k+r}$ such that $D(i_1) = D(i_2) = \ldots = D(i_{k+r}) = \lambda$.*

Proof 4.2 *Suppose A has a clustered eigenvalue λ of size $2r + k$ with $k > 0$, then $dim(NULL(A - \lambda)) = 2r + k$. Since $rank(U) = r, dim(NULL(U^T)) = n - r$. Then*

$$dim(NULL(A - \lambda) \cap NULL(U^T)) = r + k.$$

Suppose $x_1, x_2, \ldots, x_{r+k}$ is a basis for the intersection. Then:

$$Ax_i = Dx_i = \lambda x_i, \quad i = 1, 2, \ldots, r + k.$$

This means $Dx_i(j) = \lambda x_i(j), \forall j = 1, 2, \ldots, n$. Since $x_1, x_2, \ldots, x_{r+k}$ are linearly independent, then there exists at least $r + k$ distinct non-zero entries $l_1 < l_2 < \ldots < l_{r+k}$ such that:

$$Dx_{l_m}(j_{l_m}) = \lambda x_{l_m}(j_{l_m}), \quad m = 1, 2, \ldots, r + k,$$

which implies:

$$D(j_{l_1}) = D(j_{l_2}) = \ldots = D(j_{l_{r+k}}).$$

The above proposition shows that a large eigenvalue cluster in A leads to at least an entry in D repeating more than r times. Then by the first condition of deflation(ref 4.1), we can deflate an eigenpair of A. Applying this idea recursively, we can conclude that no eigenvalue clusters of size larger than $2r$ can exist for a deflated input matrix A. Since the deflation step does only sorting and without any flops, it puts a moderate overhead of at most $O(n \log n)$ (*i.e.* sorting) to the overall complexity.

4.2 LDL^T decomposition

LDL^T decomposition is an variant of LU decomposition for symmetric matrix:

$$A - \hat{\lambda} I = L\hat{D}L^T,$$

where $L \in \mathbb{R}^{n \times n}$ is a lower triangular matrix with unit diagonals, and $\hat{D}$ is a diagonal matrix. This factorization comes directly from LDU decomposition where the factors L and U are forced to have unit diagonals. Since A is symmetric, $A = LDU = U^T D L^T$. By the uniqueness of LU decomposition, $U = L^T$ and $A = LDL^T$. This factorization can be obtained by doing Gaussian elimination, then extract the diagonal entries of D from U. However, this approach does not take advantage of symmetry. An efficient algorithm should avoid calculating and storing the upper triangular part of L at all and only consider the non-trivial entries. Below is an example of implementation provided by Golub and Van Loan [25, Chapter 4.2.1].

Algorithm 4.2.1 LDL^T decomposition for a symmetric matrix

1: **procedure** $LDL^T(A, n)$ ▷ Output a lower triangular factor L and an array d storing the diagonal entries of D
2: **for** j=1:n **do**
3: **for** i = 1:j-1 **do**
4: $v(i) = L(j,i)d(i)$
5: **end for**
6: $v(j) = A(j,j) - \sum_{i=1}^{j-1} L(j,i)v(i)$
7: $d(j) = v(j)$
8: $L(j+1:n, j) = (A(j+1:n, j) - \sum_{i=1}^{j-1} L(j+1:n, i)v(i))/v(j)$
9: **end for**
10: **end procedure**

In the above implementation, if $v(j)$ gets too close or equal to zero in line 7, then A is singular or near singular and does not permit a LU decomposition. When using LDL^T for solving the shifted linear system, $A - \hat{\lambda}$ can get near singular when $\hat{\lambda}$ approaches a real eigenvalue. In the implementation of our decomposition algorithm, we use a small threshold value with the same sign as $v(j)$ to prevent overflow in the next step.

If A is symmetric positive definite, D factor is also positive definite and the factorization can be computed in a stable and efficient way. However, when A is indefinite, the D factor becomes indefinite as well and LDL^T decomposition is not numerically stable. Furthermore, such factorization may not even exist for certain non-singular matrices A, *e.g.* matrices with zeros on the diagonal.

Pivoting strategies are usually adopted to overcome such issues. Partial pivoting is the most commonly adopted pivoting strategy for LU decomposition. In every iteration, it picks the entry with largest absolute value in a column and swap it with the current pivot element *i.e.* $A(i,i)$. This can control element growth to some extent in the trailing matrix after one iteration. However, there are known test matrices [29] for which this pivoting strategy results in very large entries in the U factor, which translates to large entries in the D factor for LDL^T decomposition, which leads to loss of accuracy. Complete pivoting guarantees a non-exponential element growth rate and can thus deliver stability for all matrices. However, it requires examining all remaining

entries for every iteration and puts a $O(n^3)$ overhead to the factorization algorithm. There is extensive literature on pivoting strategies specifically for LDL^T decomposition, such as diagonal pivoting proposed in [11], which is more similar to complete pivoting in Gaussian Elimination. This strategy is employed in LAPACK implementations for solving indefinite symmetric linear systems and the stability is proved in [38].

Since the entries of $A - \hat{\lambda}I = D - \hat{\lambda}I + UHU^T$ are not directly given, we do not adopt any pivoting strategy as they require pre-computation of all entries for later comparisons. Instead, we propose a method that modifies Algorithm 4.2.1 and processes A column by column to improve efficiency. We also try to optimize storage by avoiding computation and storage of the L factor *i.e.* the sums $\sum_{i=1}^{j-1} L(j,i)v(i)$, $\sum_{i=1}^{j-1} L(j+1:n,i)v(i)$. In addition, linear systems can be solved by forward and backward substitutions with L and the number of eigenvalues smaller than $\hat{\lambda}$ is equal to the number of negative entries in the D factor by the law of Inertia:

Theorem 4.1 (Inertia) *The inertia of a symmetric matrix is the triple Inertia$(A) \equiv (\gamma, \psi, \pi)$, where γ is the number of negative eigenvalues of A, ψ is the number of zero eigenvalues of A and π is the number of positive eigenvalues of A.*
Let X be a non-singular matrix, then A and XAX^T have the same inertia. [18]

In the actual implementation of the decomposition method, we try to optimize for storage by avoiding the actual computation of the L factor and thus avoid the need to store it. Instead, we use an auxiliary matrix $G \in \mathbb{R}^{r \times n}$, the ith column of which stores a vector that can be used to recover the ith column of L later. This leads to $O(nr)$ extra storage instead of the $O(n^2)$ needed to store the L factor or the lower triangular part L.

The blocked structure for one iteration of our decomposition method is:

$$\begin{bmatrix} 1 & 0 \\ l & L_{22} \end{bmatrix} * \begin{bmatrix} d_1 & \\ & \hat{D}_{22} \end{bmatrix} * \begin{bmatrix} 1 & l^T \\ 0 & L_{22}^T \end{bmatrix} = \begin{bmatrix} a_{11} & A_{21}^T \\ A_{21} & A_{22} \end{bmatrix}.$$

$$\begin{bmatrix} d_1 & d_1 l^T \\ d_1 l & d_1 l l^T + L_{22}\hat{D}_{22}L_{22}^T \end{bmatrix} = \begin{bmatrix} a_{11} & A_{21}^T \\ A_{21} & A_{22} \end{bmatrix}.$$

The above equations give the update formulas:

$$\begin{cases} d_1 & = a_{11} \\ d_1 l & = A_{21} \\ L_{22}\hat{D}_{22}L_{22}^T & = A_{22} - d_1 l l^T. \end{cases}$$

Putting U and H back into the update formula:

$$l = U(2:n,:)HU^T(:,1)/d_1;$$

Let $g = HU^T(:,1)$ and the trailing matrix update becomes:

$$\begin{aligned}\hat{A}_{22} &= A_{22} - \frac{1}{d_1}U(2:n,:)HU^T(:,1)U(1,:)H^TU^T(2:n,:) \\ &= U(2:n,:)(H - \frac{1}{d_1}HU^T(:,1)(HU^T(:,1))^T)U^T(2:n,:) + D_{22} \\ &= U(2:n,:)(H - \frac{1}{d_1}gg^T)U^T(2:n,:) + D_{22}\end{aligned}$$

This equation shows that the trailing matrix $\hat{A}_{22}$ has a similar structure as the original problem. Hence, we establish a recursive scheme that does a rank-one update of H in every iteration and stores vector g as columns of the auxiliary matrix G for later use in forward substitutions for solving the linear system. Restoration of the L factor and the rank-one update done in our decomposition are as follows:

$$l = U(2:n,:) * g/d_1, \quad \hat{H} = H - \frac{1}{d_1}gg^T.$$

Here is the pseudo code of the above algorithm:

Algorithm 4.2.2 LDL^T algorithm on D, U, H

Require: $H = H^T$
  procedure DEIGLDL($D, U, H, \hat{\lambda}, \hat{D}, G$, info)
    for i=1,n **do**
      $G(:,i) \leftarrow H * U(:,i)$
      $\hat{D}(i) \leftarrow D(i) - \hat{\lambda} + G(:,i)^T * U(:,i)$
      if $\hat{D}(i) < \epsilon$ **then**
        $\hat{D}(i) \leftarrow \hat{D}(i) > 0?\epsilon : -\epsilon$
        if i == n **then**
          info = 0
        else
          info = 2
        end if
      end if
      $G(:,i) \leftarrow G(:,i)/\hat{D}(i)$
      $H \leftarrow H - \hat{D}(i) * G(:,i) * G(:,i)^T$
    end for
  end procedure

In each iteration, the first step to compute $HU(:,i)$ takes $2r^2$ flops and the last step of rank-one update on H also costs $2r^2$. Hence the total number of flops done by this method in one iteration is $O(4r^2)$, which accumulates to $O(4nr^2)$ in n iterations. The

number of memory accesses for each iteration is $2r$, *i.e.* reading a column from matrix U and writing a column to matrix G. Hence the total number of memory access is $O(nr)$ and this is the lower bound we can possibly achieve as processing the matrix U takes $O(nr)$ memory accesses. Hence, the computational intensity(flops / memory access) of Algorithm 4.2.2 is $O(r)$.

As shown in Algorithm 4.2.2, a threshold value is used to prevent overflow when entries in $\hat{D}$ gets close to zero. This happens more often when the shift $\hat{\lambda}$ approaches an exact eigenvalue of A and the shifted matrix $A - \hat{\lambda}I$ becomes near-singular. Although it can be shown that this method is numerically stable in the tridiagonal case [18], it suffers from the same instability issues as the LU decomposition in our case. Given reasons mentioned earlier in this chapter, we do not adopt any pivoting strategy for numerical stability. Alternatively, we provide a backup stable decomposition method when numerical instability happens. The details are presented in chapter 5.

4.3 Forward substitution

We conclude this chapter by presenting the actual method used for forward substitutions with the auxiliary storage matrix G introduced in Algorithm 4.2.2.Solving linear system $L\hat{D}L^T x = b$ with outputs of the Algorithm 4.2.2 is done in three steps:

1. Forward solve $Ly = b$;
2. Scaling $y \leftarrow y/\hat{D}$;
3. Backward solve $L^T x = y$.

Based on the equations mentioned in chapter 4.2, the matrix L can be restored by:

$$L(i,j) = U(i,:) \cdot G(:,j), \quad \forall 2 \le i \le n, 1 \le j \le i-1.$$

The ith equation of the forward solve:

$$y(i) = b(i) - \sum_{j=1}^{i-1} L(i,j) * y(j) \tag{4.1}$$

$$= b(i) - U(i,:) * \sum_{j=1}^{i-1} G(:,j)y(j). \tag{4.2}$$

Noticing that the ith entry of the solution: $y(i)$ is only calculated from $b(i), U(i,:)$ and $\sum_{j=1}^{i-1} G(:,j)y(j)$, we can optimize the method by accumulating $G(:,j)y(j)$ after solving for the jth entry of y. Then every entry $y(i)$ requires $O(r)$ flops to solve, *i.e.* the inner product between $U(i,:)$ and the accumulation $\sum_{j=1}^{i-1} G(:,j)y(j)$. Hence the

total flops done by this algorithm is $O(nr)$ and we use extra storage of a length r vector to hold the accumulation sum $\sum_{j=1}^{i-1} G(:,j)y(j)$.

Algorithm 4.3.1 Solving the linear system $Lx = b$ or $L^T x = b$

---

procedure DEIGTRS(U, G, b, opt) ▷ b will be overwritten by the solution
  temp = zeros(n, 1)
  if opt == 'N' or 'n' **then**
    for i = 2, n **do**
      temp = temp + $b(i-1) * G(:, i-1)$
      $b(i) = b(i) - U(i,:)*$ temp
    end for
  else
    for i = n-1, 1, -1 **do**
      $b(i) = b(i) - G(:,i)*$ temp
      temp = temp + $U(:,i) * b(i)$
    end for
  end if
end procedure
function DEIGLNS($\hat{D}, U, G, b$) ▷ b will be overwritten by the solution
  DEIGTRS(U, G, b, 'N')
  for i = 1, n **do**
    $b(i) = b(i)/\hat{D}(i)$
  end for
  DEIGTRS(U, G, b, 'T')
  return b
end function

Chapter 5

Numerically stable methods

In this chapter, we include the details of the methods mentioned in previous chapters for numerical stability. We begin by introducing some background information about QR decomposition in chapter 5.1. Then we move on to discuss in chapter 5.2 how Householder transformations are used to efficiently transform the shifted linear system into an upper triangular one and the stability comes directly from orthogonal transformations. To enable Householder transformations, an auxiliary variable is needed to transform the system into a slightly larger one and we will justify the efficiency given the larger problem size. Due to increased flops done and less favorable data access patterns, this Householder based QR decomposition has a higher computational cost. As mentioned in chapter 4.2, it will only be used when Algorithm 4.2.2 encounters numerical instability, which happens pretty rarely.

As mentioned in chapter 3.3, one of the difficulties for symmetric eigenvalue problem is the loss of orthogonality of eigenvectors corresponding to eigenvalue clusters. The reason is briefly mentioned in chapter 3.3 and will be revisited in more detail in chapter 6. In section 5.3, we introduce a method based on eigenvalue deflation [50, Chapter 4] that computes numerically orthogonal eigenvectors given one initial eigenvector and an eigenvalue cluster. To clarify the abuse of the word "deflation", eigenvalue deflation is different than the deflation pre-process mentioned in chapter 4.1. It is often used in iterative algorithms to find the rest of eigenvectors given one computed eigenpair, such as the restarting part in Arnoldi algorithm [61]. In our approach, we aim to "deflate" an eigenpair to get a smaller matrix with the same low-rank structure, whose eigenvectors corresponding to the eigenvalue cluster are orthogonal to the given eigenvector. The eigenvectors of deflated matrix are then computed by inverse iteration with the computed eigenvalue as shift and accumulated to eigenvectors of the original matrix by multiplying orthogonal matrices.

To conclude this chapter, we discuss our strategy for handling various situations when the bracketing algorithm converges to an eigenpair, or an eigenvalue may it be from bisection. Unlike usual bracketing algorithm, our generated eigenvalue estimates are not guaranteed to stay inside the intervals. In addition, a converged eigenpair can

come from either Rayleigh Quotient Iteration or bisection and they require different handling techniques when dividing intervals for the bracketing algorithm. The details are discussed in chapter 5.4.

5.1 Background on QR decomposition

Although it is shown that the LDL^T decomposition is stable in the tridiagonal case [18], it still suffers from the same instability as LU decomposition in our banded case. Hence, we propose a stable method using QR decomposition to solve the shifted linear equation. QR decomposition is one of the fundamental problems in numerical linear algebra. It factorizes a general $m-$by$-n$ matrix A into two factors Q and R, where Q is a column orthogonal matrix with $Q^TQ = I$ and R is an upper triangular matrix. The QR decomposition is most well-known as a solution method to linear least squares problem. In the eigenproblem domain, it is also used in the QR algorithm for symmetric eigenproblem [18, Chapter 5] and remains one of the top choices nowadays.

The QR decomposition was first proposed as the matrix form of the Gram-Schmidt process to compute orthogonal basis of a matrix's column space. A modified version "Modified Gram-Schmidt" soon came out since the original version turned out to be numerically unstable. Although Modified Gram-Schmidt process offers fairly good numerical properties, it consists of operations that are not favorable for high performance, *i.e.* the computation of each element of the triangular matrix R requires large vector inner products and many synchronizations [33]. Blocking methods exist to help with data reuse in which the matrix to be factored is first partitioned into smaller submatrices and then independently factored [33].However, such methods turned out to be subject to loss of orthogonality in the Q factor [7].

Two other ways to compute the QR decomposition remain popular choices in most of the implementations of current software packages. First one is the Givens Rotation QR decomposition [24]. In Givens Rotation method, a sequence of orthogonal transformations is applied to the input matrix A that place zeros in the trapezoidal submatrix below the main diagonal. Each orthogonal transformation, denoted by $G(\theta)$ is a plane rotation called Givens Rotation. For example in the two-dimensional case, a unitary matrix is chosen so that:

$$G(\theta)\begin{bmatrix} f \\ g \end{bmatrix} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}\begin{bmatrix} f \\ g \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix}$$

The Givens Rotation algorithm usually starts from the bottom two rows of matrix A in the left-most two columns. These entries of A determine the first Givens Rotation $G_{m,1}(\theta)$ that will place a zero in the left bottom corner of matrix A and be multiplied to $A(m-1:m,:)$. Then $G_{m,1}(\theta)^T$ may be multiplied to $Q(:,m-1:m)$ in place to accumulate the Q factor. Then the algorithm moves up one row to generate $G_{m-1,1}(\theta)$, which will be applied to $A(m-2:m-1,:)$ and $Q(:,m-2:m-1)$. This proceeds until

the main diagonal so that the whole column is zeroed out below the diagonal. Then the algorithm moves along the columns to zero out the rest of the lower triangular part of matrix A. Due to its nature of introducing zeros one at a time, this algorithm is more favorable in transforming input matrices that are almost upper triangular.

The other popular method to compute QR decomposition is based on Householder transformations. This algorithm transforms the input matrix A to an upper triangular matrix R by applying a sequence of *Householder reflections* [25]. A *Householder reflection* is an orthogonal transformation in the form of:

$$P = I - \beta h h^T,$$

where h is called a *Householder vector*. To alleviate cancellation effect in numerical computation, h is usually chosen based on input vector x such that $Px = ||x||e_1$ [18], where P is the unitary transformation matrix and e_1 is the first column of the identity matrix. In the kth iteration, the vector $x_k = A(k : m, :)$ is selected as an input to compute the Householder reflector P_k for this iteration, which will then be applied to $A(k : m, k + 1 : n)$. This will zero out all entries of the kth column of A below the diagonal.

$$\begin{bmatrix} * & \times & \times & \times \\ * & \times & \times & \times \\ * & \times & \times & \times \\ * & \times & \times & \times \end{bmatrix} \leftarrow \begin{bmatrix} \times & \times & \times & \times \\ 0 & * & \times & \times \\ 0 & * & \times & \times \\ 0 & * & \times & \times \end{bmatrix} \leftarrow \begin{bmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & * & \times \\ 0 & 0 & * & \times \end{bmatrix} \leftarrow \begin{bmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times \end{bmatrix}$$

The above equation is an illustration of *Householder* QR decomposition, where $*$ represents the vector used to generate *Householder reflectors* in each iteration. This algorithm zeros out one column per iteration and stops until the input matrix is transformed into an upper triangular one. The invariant $A = (P_1^T P_2^T \dots P_n^T)(P_1 P_2 \dots P_n A)$ is maintained the the QR decomposition is given by:

$$Q = P_1^T P_2^T \dots P_n^T$$
$$R = P_1 P_2 \dots P_n A$$

Another reason that this algorithm is a favorable choice for QR decomposition is that the multiplication by *Householder reflectors* does not necessarily need to be done by matrix multiplication, *e.g.*

$$PA = (I - \beta h h^T)A = A - h w^T,$$

where $w = \beta A^T v$. The following gives a basic implementation of Householder QR:

The function $House(x)$ returns the *Householder vector* h and the corresponding coefficient β such that $(I - \beta h h^T)x = ||x||e_1$. Although the above algorithm is conceptually simple in implementation, it is dominated by matrix-vector multiplications.

Algorithm 5.1.1 Householder based QR decomposition

procedure QR DECOMPOSITION(A, m, n)
  for i = 1:n **do**
    [h, β] = House(A(i:m, i))
    $A(k:m, k:n) = A(k:m, k:n) - \beta vv^T A(k:m, k:n)$
    $Q(1:m, k:m) = Q(1:m, k:m) - \beta Q(1:m, k:m)vv^T$
  end for
end procedure

Such computations have very poor data re-usability and lead to high communication costs. Blocking can be used to improve the performance of Householder QR where instead of multiplying the *Householder reflectors* as rank-one updates of the identity matrix, delay the application of several *Householder reflections* as matrix multiplications. Bischof and Van Loan [7] explains a way to delay the application of *Householder reflectors* in blocks using matrix-matrix multiplications. *i.e.* use $m - by - b$ matrices W and Y to represent the first b *Householder reflections*:

$$\begin{aligned} P_{wy} &= P_1 P_2 \dots P_b \\ &= I + WY^T \end{aligned}$$

And the application of P_{wy} to A can be done with $P_{wy}A = A + W(Y^T A)$. This update requires only $O(mnb)$ flops, which is fewer than the flops required by multiplying P_{wy} and A directly. The above operations are both rich in matrix-matrix multiplications and can be expected to achieve high performance in modern computer architectures. The block size b can be chosen based on the memory capacity of the target architecture. Algorithm 5.1.1 may be modified by partitioning the columns of A into blocks of b columns, for each column block, b *Householder reflectors* are computed and the corresponding W, Y are generated. Then, instead of applying *Householder reflections*, the matrix form $P_{wy} = I + WY^T$ is applied to both Q and the trailing part of A as mentioned above [7].

The choice of our stable decomposition method ends up being Householder QR for the convenience of zeroing out a whole column per iteration. As indicated in the above algorithm, the main computational costs for Householder QR come from updating the trailing matrix, which accumulates to $O(nm^2)$ flops for a $m \times n$ matrix A. In the next section, we will further discuss how to adapt Householder QR in our case so that the overall flops done remains $O(nr^2)$ .Although this method is more expensive than Algorithm 4.2.2 due to Householder transformations, it is mostly avoided if the backward error $\frac{||b-Ax||}{||x||}$ of the solution given by LDL^T is small enough.

5.2 QR variant of decomposition method

In this section we discuss the detailed formation of our QR decomposition method, which is based on Householder QR, for solving the shifted linear system. For simplicity, we merge the shift $D - \hat{\lambda}I$ as D in the following discussions of this chapter. Suppose the system that needs to be solved is $(D + UHU^T)x = b$, define an auxiliary variable $y = HU^T x$ to transform the system:

$$\begin{cases} D + Uy & = b \\ HU^T x - y & = 0 \end{cases} \tag{5.1}$$

Putting it in matrix form:

$$\begin{bmatrix} HU^T & -I \\ D & U \end{bmatrix} * \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix}.$$

Let

$$\hat{A} = \begin{bmatrix} HU^T & -I \\ D & U \end{bmatrix}, \quad \hat{x} = \begin{bmatrix} x \\ y \end{bmatrix}, \quad \hat{b} = \begin{bmatrix} b \\ 0 \end{bmatrix}, \quad \hat{W} = \begin{bmatrix} -I \\ U \end{bmatrix}$$

The new system becomes $\hat{A}\hat{x} = \hat{b}$, with a slightly larger matrix $\hat{A} \in \mathbb{R}^{(n+r)\times(n+r)}$. Since H is an $r \times r$ matrix, there are r non-zero entries below the diagonal of the first n columns of $\hat{A}$. Householder vectors of length $r + 1$ can be used to zero-out the off-diagonal entries. i.e. in the ith iteration, a Householder vector zeros out $\hat{A}(i : i+r, i)$ because the first $i-1$ rows are already modified in the previous iterations. The transformation has the following shapes:

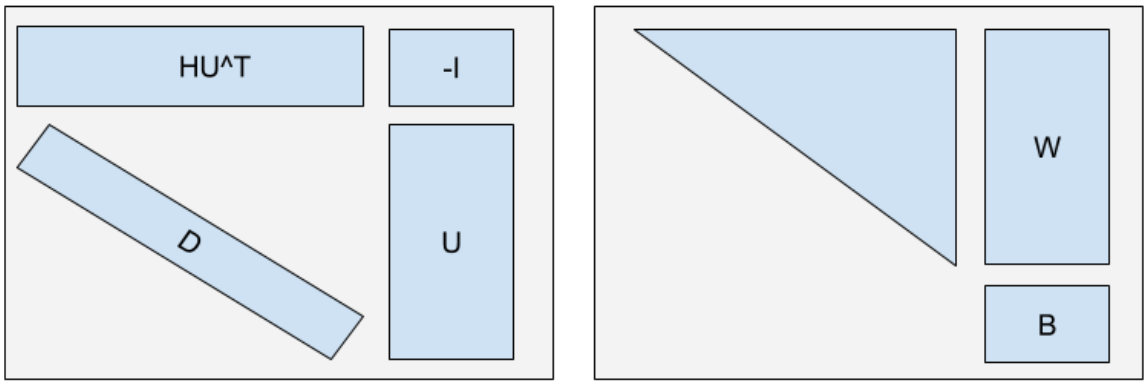

Figure 5.1: Shapes for QR transformation

Let the $\hat{A}^{(i)}, \hat{W}^{(i)}$ be the matrices obtained by applying Householder transformations to $\hat{A}, \hat{W}$ after $i - 1$ iterations respectively. Then another Householder vector h is used to zero-out the off-diagonals of $\hat{A}^{(i)}(i : i + r, i)$, and applied to the trailing matrix:

$$\hat{A}^{(i)}(i : i + r, i : n + r) = [\hat{A}^{(i)}(i : i + r, i : n) \quad \hat{W}^{(i)}(i : i + r, :)]$$

Since the right part $\hat{W}^{(i)}(i : i+r, :)$ is a small $(r+1) \times r$ block, the main cost of Householder transformation is on the left part $\hat{A}^{(i)}(i : i+r, i : n)$ of the trailing matrix.The following proposition shows that only a small part of the Householder transformation matrix needs to be kept:

Proposition 5.1 *Let V_i be the Householder matrix $I - \tau hh^T$ in the ith iteration and $V^{(i)} = V_i V_{i-1} \dots V_1$ be the accumulation of i such matrices. Then $V^{(i)}$ has the following two properties:*

1. *In the ith iteration, $V^{(i-1)}(i+r : n+r, i+r : n+r) = I$;*
2. *In the ith iteration, only $V^{(i-1)}(i : i+r-1, 1 : r)$ is multiplied to $\hat{A}^{(i)}$ for the next Householder vector.*

Proof 5.1 *The first property is true because the ith Householder vector $h_i = [\hat{h}_i \quad 0]^T$ has $n-r-i$ trailing zeros. So $V_i(i : end, i : end)$ has the shape:*

$$\begin{bmatrix} h_i h_i^T & 0 \\ 0 & I_{n-i+1} \end{bmatrix}$$

and $V^{(i)} = V_i * V^{(i-1)}$ will also have property 1 by induction.

For the second property, we have:

$$\hat{A}^{(i)}(i : i+r, i) = V^{(i-1)} \hat{A}(i : i+r, i).$$

Rewriting into block matrix form:

$$\begin{aligned} \hat{A}^{(i)}(i : i+r, i) &= V^{(i-1)}(i : i+r, 1 : r+i) * \begin{bmatrix} HU^T(:, i) \\ 0 \\ \vdots \\ D(i) \end{bmatrix} \\ &= \begin{bmatrix} V^{(i-1)}(i : i+r-1, 1 : i+r-1) & 0 \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} HU^T(:, i) \\ D(i) \end{bmatrix} \text{ (by property 1)} \\ &= \begin{bmatrix} V^{(i-1)}(i : i+r-1, 1 : r) * HU^T(:, i) \\ D(i) \end{bmatrix} \end{aligned}$$

Hence only $V^{(i-1)}(i : i+r-1, 1 : r)$ and $D(i)$ are used for next Householder vector. This property means that only $r+1$ rows of the Householder matrix needs to be updated every iteration, i.e $V^{(i-1)}(i : i+r, 1 : r)$. Also, since only the first r columns of the $V^{(i)}$ matrix is used, we use an $(n+r) \times r$ matrix to store this information.

Let $\tilde{b}$ denote the vector b after n Householder transformations, then:

$$\begin{cases} \hat{A}^{(n)}(1:n,1:n)x + \hat{W}^{(n)}(1:n,:)y & = \tilde{b}(1:n) \\ \hat{W}^{(n)}(n+1:n+r,:)y & = \tilde{b}(n+1:n+r) \end{cases} \tag{5.2}$$

The second equation in 5.2 is solved by QR decomposition on $\tilde{W}(n+1:n+r,:)$, with $O(r^3)$ flops. Then y is substituted into the first equation, leaving a triangular system of size $n \times n$:

$$\hat{A}^{(n)}(1:n,1:n)x = \tilde{b}(1:n) - \hat{W}^{(n)}(1:n,:)y.$$

Since this system has the same structure as the previous system obtained from Algorithm 4.2.2 in that the upper triangular factor $\hat{A}^{(n)}(1:n,1:n)$ can be formed by $V^T * HU$, it can be solved with the previously mentioned Algorithm 4.3.1.

Algorithm 5.2.1 QR algorithm for linear solving

Require: *House* is a function to generate householder vector

procedure DEIGSQR($D, U, H, \hat{\lambda}, x, b$)

$V = [I_r \quad 0]$

$W = [-I_r \quad U^T]$

$\hat{b} = [0 \quad b]$

$HU = H * U^T$

for i = 1, n **do**

$h = \text{HOUSE}([V(:, i:i+r-1) * HU(:,i) \quad D(i) - \hat{\lambda}]^T)$

$\hat{D}(i) = h(1)$

$\hat{V} = I - 2hh^T$

$W(i:i+r,:) = \hat{V} * W(i:i+r,:)$

$\hat{b}(i:i+r) = \hat{V} * \hat{b}(i:i+r)$

$V(:, i:i+r) = V(:, i:i+r) * \hat{V}$

end for

$W(:, n+1:r+n) = QR$ ▷ Solve the last block equation

$W(:, n+1:n+r)^T y = \hat{b}(n+1:n+r)$

$y = R^{-T}\hat{b}(n+1:n+r)$

$y = Qy$

$x = \hat{b}(1:n) - W(:,1:n)^T * \hat{b}(n+1:n+r)$ ▷ Solve the first equation in (1)

DEIGTRS(V^T, HU, x, 'T') ▷ Triangular solve using previous subroutine

end procedure

In every iteration, the cost for *Householder vector* generation is $O(r+1)$ and the cost for applying the transformation to the trailing part is $O((r+1) * r)$. Updating

the Householder vector matrix takes $O((r+1)*r)$ per iteration and updating the right-hand-side of the equation only costs $O(r+1)$. Hence, the accumulated flops needed by Householder transformations for Algorithm 5.2.1 is $O(nr^2)$. Transforming the last $r \times r$ block into upper triangular matrix using QR decomposition adds a $O(r^3)$ overhead to Algorithm 5.2.1 and the forward substitutions after the input A becomes upper triangular takes $O(nr)$ flops to complete, as mentioned in chapter 4.3. Thus, the overall computational cost for Algorithm 5.2.1 becomes $O(nr^2)$. However, Algorithm 5.2.1 reads and writes a $(r+1) \times r$ block to the matrix V in every iteration and hence performs $O(nr^2)$ memory accesses. This causes the QR algorithm to have a computational intensity of $O(1)$ instead of $O(r)$ compared to Algorithm 4.2.2, which adds to the inefficiency of Algorithm 5.2.1.

5.3 Orthogonal deflation and eigenvalue clusters

In usual bracketing algorithm, eigenvalues are computed first and their corresponding eigenvectors are computed later by inverse iteration. In such situations, we encounter the problem of computing the rest of eigenvalues of A given one found eigenpair (λ_1, u_1). A technique for achieving this is commonly known as a eigenvalue deflation. Typically, a rank one modification is applied to the original matrix to displace the found eigenvalue λ_1, while keeping all other eigenvalues unchanged. Usually, the rank-one modification is chosen that a special eigenvalue can be found in the next iteration. For example when using Power Iteration [18], deflation can be used so that the next eigenvalue to be found is the eigenvalue of the input matrix with second largest norm.

In the general case of deflation, known as Wielandt's deflation [50], the deflated matrix has the form:

$$\hat{A} = A - \sigma u_1 v^T,$$

where v is an arbitrary vector such that $v^T u = 1$, and σ is a shift of choice. The following theorem shows that the eigenvalues of $\hat{A}$ are the same as those of A except that λ_1 is changed to $\lambda_1 - \sigma$.

Theorem 5.1 (Wielandt) *[50] Let the eigenvalues of A be $\lambda_1, \lambda_2, \ldots, \lambda_n$, then the spectrum of $\hat{A}$ is defined by $\lambda_1 - \sigma, \lambda_2, \ldots, \lambda_n$.*

The above rank-one modification preserves the eigenvector u by

$$\hat{A}u = Au - \sigma u_1 v^T u_1 = Au_1 - \sigma u_1 = (\lambda_1 - \sigma)u_1.$$

It is important to see what the other eigenvectors have become. Suppose u_i is the eigenvector of A corresponding to λ_i. Consider the vectors of form $\hat{u}_i = u_i - \gamma_i u_1$, we have:

$$\begin{aligned}\hat{A}\hat{u}_i &= (A - \sigma u_1 v^T)(u_i - \gamma_i u_1) \\ &= \lambda_i u_i - (\gamma_i \lambda_1 + \sigma v^T u_i - \sigma\gamma_i)u_1.\end{aligned}$$

It can be seen that if γ_1 is set to 0, then $\hat{u_1} = u_1$ is an eigenvector of $\hat{A}$ corresponding to $\lambda_1 - \sigma$. For $i \neq 1$, it is possible to pick γ_i such that $\hat{u}_i$ is an eigenvector of $\hat{A}$ corresponding to λ_i [50, Chapter 4]:

$$\gamma_i \equiv \frac{v^T u_i}{1 - (\lambda_1 - \lambda_i)/\sigma}.$$

In theory, there are infinitely many possible ways to pick the vector v. For symmetric matrices, one of the most common choices is to take $v = u_1$. This is referred to as Hotelling's deflation. It can also be shown that this choice results in an almost optimal condition number for the second eigenvalue to be computed, *i.e.* for λ_2 [50, Chapter 4].

For our particular problem, we face the loss of orthogonality of eigenvectors corresponding to an eigenvalue cluster. In this case, we obtain a computed eigenpair $(\hat{\lambda}, u)$ from the bracketing algorithm and we aim to find the other eigenvectors corresponding to the eigenvalue cluster at $\hat{\lambda}$. At the same time, we require the computed eigenvectors to be numerically orthogonal. We propose an "orthogonal deflation" method for this problem due to the similarity in situation with that of the deflation problem mentioned above. Since our goal is to ensure orthogonality,orthogonal transformations are used instead of subtractions to transform A such that the eigenvalues remain unchanged and the deflated matrix has eigenvectors orthogonal to u. Formally, we aim to deflate A and generate D', U', H' such that $A' = D' + U'H'U'^T$ has the following properties:

1. $D' \in \mathbb{R}^{(n-1)\times(n-1)}$;
2. A' has all eigenpairs of $D + UHU^T$ except for $(\hat{\lambda}, u)$;
3. Eigenvectors of A' corresponding to $\hat{\lambda}$ are orthogonal to u.

Suppose $q \perp u$ is an unknown eigenvector orthogonal to u. Consider the Householder transformation matrix V such that $Vu = (\beta, 0, 0, \ldots, 0)^T$, since $q \perp u$ and V is orthogonal, $Vu \perp Vq$.So if $\beta \neq 0$, then $Vq(1) = 0$(which will always be the case because $\beta = ||u|| > 0$).In other words, $Vq = (0, q')^T$ for a length $n-1$ vector q'.

Let $A = Q\hat{D}Q^T$ where $Q(:, 1) = u$ and $\hat{D}(1, 1) = \hat{\lambda}$, then

$$VQ = \begin{bmatrix}\beta & 0 \\ 0 & Q'\end{bmatrix}, \quad B = VAV^T = \begin{bmatrix}\beta & 0 \\ 0 & Q'\end{bmatrix}\begin{bmatrix}d_1 & 0 \\ 0 & D'\end{bmatrix}\begin{bmatrix}\beta & 0 \\ 0 & Q'^T\end{bmatrix} = \begin{bmatrix}\beta d_1 & 0 \\ 0 & Q'D'Q'^T\end{bmatrix}$$

Hence Q' is the matrix whose columns are eigenvectors for $B(2:n, 2:n)$. Let $V = I - 2hh^T$ and $h = (h_1, h_2)^T$ where $h_2 \in \mathbb{R}^{n-1}$, writing B in blocked form:

$$B = \begin{bmatrix} 1-h_1^2 & -2h_1h_2^T \\ -2h_1h_2 & I - 2h_2h_2^T \end{bmatrix} \begin{bmatrix} A_{11} & A_{21}^T \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} 1-h_1^2 & -2h_1h_2^T \\ -2h_1h_2 & I - 2h_2h_2^T \end{bmatrix}$$

and:

$$B(2:n, 2:n) = A_{22} + \tau h_2 A_{21}^T + \tau A_{21} h_2^T + \rho h_2 h_2^T \tag{5.3}$$

with some unknown constants τ and ρ. In our choice of Householder reflections with coefficient being 2, the constants are:

$$\tau = -\sqrt{\frac{2}{1+|u_1|}}, \quad \rho = (A_{11} - \lambda)\frac{2}{1+|u_1|}.$$

Since $A = D + UHU^T$, the blocks of A have the following structure:

$$A_{22} = D(2:n) + U(2:n,:)HU^T(2:n,:), \quad A_{21} = U(2:n,:)HU^T(1,:)$$

Rewriting 5.3 with the above equation:

$$\begin{aligned} B(2:n, 2:n) = D(2:n) + U(2:n,:)HU^T(2:n,:) + \\ \tau h_2 U(2:n,:)HU^T(1,:) + \tau U(1,:)HU^T(2:n,:)h_2^T + x h_2 h_2^T, \end{aligned}$$

the trailing matrix B has the form $D' + U'H'U'^T$:

$$B(2:n, 2:n) = D(2:n) + \begin{bmatrix} U(2:n,:) & h_2 \end{bmatrix} \begin{bmatrix} H & \tau HU^T(1,:) \\ \tau U(1,:)H & u \end{bmatrix} \begin{bmatrix} U^T(2:n,:) \\ h_2^T \end{bmatrix}$$

This deflation technique is used to find eigenvectors in the following steps:

1. Start with a computed eigenpair $(\hat{\lambda}, u)$ and cluster size k;
2. Deflate the eigenpair $(\hat{\lambda}, u)$ and get D', U', H'. Then find an eigenvector of $D' + U'H'U'^T$ corresponding to $\hat{\lambda}$;
3. Repeat the above process $k-1$ times;
4. Apply the householder (V) transformations from the last vector backwards to the first.

Algorithm 5.3.1 Orthogonal deflation

Require: $||u|| = 1$

function OrthogDeflate($D, U, H, \hat{\lambda}, u$) ▷ Generate Householder vector h

alpha = 1.0 + abs(u(1))

norm = 1.0 - u(1)*u(1)

h = sgn(u(1))* u(2:n) ▷ Normalize h

h = h / (norm + $\alpha * \alpha$)

▷ Generate D', U', H'

tau = -SQRT(2.0/α)

hu = $HU^T(1,:)$

γ = D(1) - $\hat{\lambda}$ + DotProduct(U(1,:), hu)

hu = tau * hu, x = γ * 2.0/α

$H0 = \begin{bmatrix} H & hu^T \\ hu & u \end{bmatrix}$, $U0 = \begin{bmatrix} U(2:n,:) & h \end{bmatrix}$, $D0 = D(2:n)$

return $H0, U0, D0$, h, α

end function

The following algorithms shows how the above orthogonal deflation can be applied to compute eigenvectors corresponding to an eigenvalue cluster.

Algorithm 5.3.2 Algorithm dealing with eigenvalue clusters

Require: $\hat{\lambda}$ is an eigenvalue approximation of A

procedure EigenMultiple($D, U, H, \hat{\lambda}, k, Q$)

$Q = 0$

x =RANDOM(n)

InverseIter($D, U, H, \hat{\lambda}, x$)

$Q(:, 1) = x$

$H_0, U_0, D_0 = H, U, D$

for i = 1, k-1 **do**

H_0, U_0, D_0, h, alpha = OrthogDeflate($D_0, U_0, H_0, \hat{\lambda}, x$)

x =RANDOM(n-i+1)

$House(i+1:n, i) = h$, $House(i, i)$ =alpha

InverseIter($D0, U0, H0, \hat{\lambda}, x$)

$Q(i:n:, i) = x$

end for

for i = k-1, 1, -1 **do**

$Q(i:n, i+1) = (I - 2House(i:n, i)House(i:n, i)^T)Q(i:n, i+1)$

end for

end procedure

In the above algorithm, *InverseIter* is the inverse iteration to find an eigenvector

given an approximated eigenvalue. The details of this method is presented in chapter 6.2. Since we use the same decomposition methods (Algorithm 4.2.2 and 5.2.1) to solve the shifted linear system within inverse iteration and it usually converges very fast (ref chapter 6.2), the complexity for one inverse iteration is $O(nr^2)$. Hence, the total cost in the middle loop for all deflated eigenvectors is $O(knr^2)$. In the last step, we accumulate the eigenvectors by applying all the Householder transformations to them. Since each Householder transformation costs $O(nr^2)$ flops in this case, the overall complexity is $O(knr^2)$ for this step.

The complexity of Algorithm 5.3.2 heavily depends on the cluster size k. As previously mentioned by Proposition 4.2, $k <= 2r$, which keeps the overall complexity in range of $O(nr^2)$.

5.4 Handling convergence

In usual bracketing algorithm, only eigenvalues are desired and when an interval containing an eigenvalue gets too narrow, convergence is declared and that interval is removed from the *Worklist*. Eigenvector computations are dealt with by an separate algorithm. However, in order to improve the overall efficiency of bracketing, we generate an a sequence of eigenpair estimates (μ_k, x_k) using a combination of Rayleigh Quotient Iteration and bisection. Thus we have two possible convergence situations:

- (μ_k, x_k) converges by Rayleigh Quotient Iteration criteria mentioned in theorem 3.1. *i.e.*
$$||(A - \mu_k I)x_k||_2 < \tau,$$
where τ is a pre-defined threshold value.
- μ_k converges by Bisection criteria. *i.e.* the interval containing μ_k gets too narrow.

For the bracketing algorithm to work, an interval containing the computed eigenvalue μ_k needs to be removed from the *Worklist*. We analyze this removal step in the following situations:

1. (μ_k, x_k) converged by Rayleigh Quotient Iteration and the interval containing μ_k contains only one eigenvalue;
2. (μ_k, x_k) converged by Rayleigh Quotient Iteration but the interval containing μ_k contains other eigenvalues;
3. μ_k is found in an interval whose length is smaller than 2τ.

For the first case, the interval containing μ_k is removed. In the third case, the interval is removed and the size of the eigenvalue cluster is determined by the *Count* of the two end points. If the cluster size is larger than one, the *Orthogonal Deflation* method mentioned previously is used to compute eigenvectors for this cluster.

For the second case, a perturbation method is used to determine a range containing the eigenvalue and creating a narrow interval. Suppose the interval containing μ from the *Worklist* is $[\alpha, \beta)$, then With τ as the desired accuracy, pick two points $\mu + \tau$ and $\mu - \tau$ in $[\alpha, \beta)$ (for edge cases just pick the one that's in the interval). Then split the interval into $[\alpha, \mu - \tau) \cup [\mu - \tau, \mu + \tau) \cup [\mu + \tau, \beta)$ and remove the narrow interval $[\mu - \tau, \mu + \tau)$. The cluster size of this eigenvalue is determined by the *Count* at $\mu - \tau$ and $\mu + \tau$.

Algorithm 5.4.1 Handle Convergence

```
1: procedure HandleConvergence(intervalList, D, U, H, mu, x, info)
2:     result = intervalList % find(mu)
3:     if result == NULL then                                  ▷ mu not found
4:         Bisection(intervalList, mu, x, D, U, H, .false., info)
5:     else if result % n == 1 then
6:         return mu, x
7:     else
8:         if result % length < τ then                         ▷ Interval too samll
9:             return mu, x
10:        else
11:            if mu - result % low < τ then
12:                Perturb(D, U, H, mu, result, intervalList, 'U', info2)
13:                info = info2 - result % nlow
14:            else if result % high - mu < τ then
15:                Perturb(D, U, H, mu, result, intervalList, 'L', info2)
16:                info = result % nhigh - info2
17:            else
18:                Perturb(D, U, H, mu, result, intervalList, 'U', info2)
19:                Perturb(D, U, H, mu, result, intervalList, 'L', info3)
20:                info = info2 - info3
21:            end if
22:        end if
23:    end if
24: end procedure
```

The *PERTURB* subroutine used above is a subroutine that perturbs μ to $\mu + \tau$ or $\mu - \tau$, computes the corresponding *Count* and inserts the rest of the interval back to the list.

Chapter 6

Error analysis for eigensystem

The solutions to an eigenvalue problem usually needs careful examination from both the accuracy and stability aspect. In addition, the errors on computed eigenvalues and eigenvectors are determined using different sets of information. For symmetric matrices, early work of Wilkinson [57] has shown that given a computed eigenpair $(\hat{\lambda}, \hat{u})$, there exists a real eigenvalue λ of A such that:

$$|\lambda - \hat{\lambda}| \leq ||A\hat{u} - \hat{\lambda}\hat{u}||.$$

This is also used as the stopping criteria for Rayleigh Quotient Iteration in our bracketing algorithm (ref 3.2). However, the above equation does not reveal any information about the error on computed eigenvector $\hat{u}$. A similar result for computed eigenvector exists but it requires information about the whole eigenspace. Suppose that the computed eigenvalues are given by $\hat{\lambda}_1, \hat{\lambda}_2, \ldots, \hat{\lambda}_n$ and the corresponding computed eigenvectors are $\hat{u}_1, \hat{u}_2, \ldots, \hat{u}_n$. Let:

$$d = \min_{i \neq j} |\hat{\lambda}_i - \hat{\lambda}_j| > 0,$$

while

$$\alpha = \max_i ||A\hat{u}_i - \hat{\lambda}_i \hat{u}_i|| > \frac{1}{2} d.$$

Then it is known from Ortega's work [40, pp:59-62] that there exists a normalized eigenvector u corresponding to an eigenvalue λ(the eigenvalue approximated by $\hat{\lambda}_1$) such that:

$$||u - \hat{u}_1|| \leq \frac{\alpha}{d - \alpha} \sqrt{1 + \frac{\alpha}{d - \alpha}}$$

For non-symmetric matrices, some methods using Gerschgorin's theorem to estimate $\hat{\lambda}_1$ and $\hat{u}_1$ exist from the work of Wilkinson [57] and Varah [55]. However, the above results except for the first one with eigenvalue are generally not available for

problems in which only a subset of the eigenpairs are computed, since they all require information from the whole computed eigensystem.

As previously mentioned in chapter 3.3, a small residual $||A\hat{u} - \hat{\lambda}\hat{u}||$ is required for both the accuracy of eigenpairs and the orthogonality of eigenvectors. In this chapter we aim to introduce the bounds on residuals of computed eigenpairs and how they are related to the orthogonality of computed eigenvectors.

Since eigenpairs are computed from two different sources by our bracketing algorithm: Rayleigh Quotient Iteration and bisection with inverse iteration, our discussion on the bounds of residuals is split into these two parts respectively. Chapter 6.1 includes the analysis of error bounds on residuals for eigenpairs found by Rayleigh Quotient Iteration. For inverse iteration, since it does not give direct implications on residual like Rayleigh Quotient, we analyze it from the perspectives of convergence, stopping criteria and finite precision arithmetic in section 6.2.1, 6.2.2 and 6.2.3. At last, we use the results from the first few sections to establish the numerical orthogonality of computed eigenvectors in section 6.3. Section 6.4 concludes the chapter by presenting the backward error on the whole computed eigensystem.

6.1 Rayleigh Quotient Iteration

To pick an appropriate stopping threshold for Rayleigh Quotient Iteration, we need to consider the size of the rounding errors when computing the residual $||(A-\hat{\lambda}I)\hat{u}||_2$ [17]. Suppose that ϵ is machine precision, and $||\hat{u}||_2 = 1$. Then the size of the rounding errors in the computation of the Rayleigh Quotient $\rho(x) = x^TAx/x^Tx$ is smaller than $n||A||_2\epsilon$. Let e denote the error when computing the residual vector $A - \rho(\hat{u})\hat{u}$, then:

$$||e||_2 \leq (n+1)\epsilon||A||_2.$$

It was observed that for symmetric matrices, convergence of Rayleigh Quotient Iteration to a real eigenvector is ultimately very rapid. In 1958 and 1959, Ostrowski [1] produced a sequence of papers in which various aspects of Rayleigh Quotient Iteration were analyzed in great details. He is also credited for the first rigorous proof of the cubic asymptotic convergence rate. Although his work applies only to starting vectors in definite small neighborhood of real eigenvectors, Kahan [8] came out with a proof that Rayleigh Quotient Iteration converges for almost all starting vectors. Due to the cubic local convergence property of the Rayleigh Quotient Iteration, once the residual size gets "considerably small", *e.g.*

$$||(A - \hat{\lambda}I)\hat{u}||_2 \leq 100n||A||_2\epsilon \tag{6.1}$$

then in the next iteration it is expected to reach round-off level. Therefore, in practice our RQI is terminated one iteration after equation (6.1) is satisfied. The extra iteration is aimed at ensuring that the residual norm reaches its round-off level.

Let τ denote the parameter $100n||A||_2\epsilon$ and $||\cdot||$ denote the 2-norm $||\cdot||_2$ in subsequent discussions.

Let $\{(\mu_k, x_k)\}$ be a sequence generated by Rayleigh Quotient Iteration and $r_k = ||(A-\mu_k I)x_k||_2$ be the residual of the kth iteration, as mentioned by theorem 3.1, there are two possibilities:

1. $\{(\mu_k, x_k)\}$ converges to an eigenpair;
2. μ_k converges to the mid point of two eigenvalues and x_k converges to a vector in an invariant space of A.

For the first case, r_k monotonically decreases and converges to zero. Our algorithm stops one iteration after $r_k \leq \tau$, making sure that the computed residual is small enough. For the second case, our algorithm checks the improvement $||r_{k+1}||/||r_k||$ and uses bisection if the improvement is not big enough. Hence even if the sequence does not converge to an eigenpair with Rayleigh Quotient Iteration, an eigenvalue is still found by bisection.

6.2 Bisection and Inverse Iteration

Inverse iteration was first introduced by Wielandt in 1944 [31] as a method for computing eigenfunctions of linear operators. It was later turned into a viable method for computing eigenvectors of matrices by Jim Wilkinson. It is still currently a common choice of method to compute eigenvectors when approximations to one of more eigenvalues are given. Inverse iteration is frequently used in structural mechanics, for example, to compute extreme eigenvalues and corresponding eigenvectors.

When an eigenvalue $\hat{\lambda}$ is found by bisection, inverse iteration is used to find the corresponding eigenvectors by the following :

$$\begin{aligned}(A - \hat{\lambda}I)z_k &= x_{k-1} \\ x_k &= z_k/||z_k||\end{aligned}$$

Again, $r_k = (A - \hat{\lambda}I)x_k$ is used to denote the residual after the kth iterate.

We would like to point out that our primary means for analyzing the convergence properties of inverse iteration is backward error rather than forward error. The backward error for an estimate x_k is the residual norm $||r_k||$. A small residual norm $||r_k||$ indicates that $(\hat{\lambda}, x_k)$ is close to an eigenpair of the matrix A. On the contrary, the forward error measures the closeness of x_k to an eigenvector. We concentrate on the backward error because it is readily available from the inverse iteration itself [32]:

$$||r_k|| = ||(A - \hat{\lambda}I)x_k|| = ||x_{k-1}/z_k|| = 1/||z_k||.$$

In the case of symmetric matrices, a measure of forward error is the acute angle θ_k between x_k and a real eigenvalue of A. Unfortunately, a small backward error does not translate directly to a small forward error. The relation and corresponding results have already been presented in chapter 3.3.

6.2.1 Convergence

For the discussion of Inverse Iteration, suppose $\hat{\lambda}$ is the computed eigenvalue, and x_0 is the starting vector. Consider the following partition of the eigensystem

$$\Lambda = \begin{bmatrix} \Lambda_1 & \\ & \Lambda_2 \end{bmatrix}, \quad Q = (Q_1, Q_2),$$

where Λ_1 contains all eigenvalues λ_i at the same, minimum distance to $\hat{\lambda}$,

$$\epsilon = ||\Lambda_1 - \hat{\lambda}I||,$$

while Λ_2 contains the remaining eigenvalues that are further away from $\hat{\lambda}$,

$$\epsilon < min_j|(\Lambda_2)_{jj} - \mu| = 1/||(\Lambda_2 - \hat{\lambda}I)^{-1}||.$$

This partition covers two common special cases: $\hat{\lambda}$ approximates an eigenvalue λ of multiplicity $l \geq 1$, *i.e.* $\Lambda_1 = \lambda I$; and $\hat{\lambda}$ approximates two distinct real eigenvalues at the same minimal distance to its left and right, *i.e.* $\hat{\lambda} - \lambda_1 = \lambda_2 - \hat{\lambda}$. The columns of Q_1 span the invariant subspace associated with the eigenvalues in Λ_1.

The following result shows that the residual norm decreases with the angle between the starting vector and the desired eigenspace [32].

Theorem 6.1 *Let inverse iteration be applied to a non-singular symmetric matrix $A - \hat{\lambda}I$ and $r_1 = (A - \hat{\lambda}I)x_1$ be the residual of the first iterate x_1. And let $0 \leq \theta \leq \pi/2$ be the acute angle between the starting vector x_0 and the eigenspace range(Q_1).*

If $\theta < \pi/2$, then:

$$||r_1|| \leq \epsilon / \cos\theta.$$

Corollary 6.1 *Under the assumption of Theorem 6.1,*

$$\epsilon \leq ||r_1|| \leq \epsilon / \cos\theta.$$

Based on these results, Wilkinson proposed that for symmetric matrices, two iterations are enough to produce a vector that's as accurate as expected [58, Section III]. The following result promises at least in exact arithmetic that for nearly eighty percent of random starting vector, one iteration is enough to drive the residual down to its minimum value:

Corollary 6.2 *In addition to the assumptions in Theorem 6.1, if θ does not exceed 75°:*

$$||r_1|| \leq 4\epsilon$$

Combining the above results, we can conclude that the number of inverse iterations needed in our algorithm is limited to 2 or 3 in most cases. The convergence of inverse iteration gets complicated when there exist eigenvalue clusters and it is one of our main use cases. The following theorem discusses this situation:

Theorem 6.2 *[32] Let inverse iteration be applied to a non-singular normal matrix $A - \hat{\lambda}I$. Given the above partition, if $\Lambda_1 = \lambda I$ for some λ and $Q_1^T x_0 \neq 0$, then the norms of the residuals $||r_k||$ decrease strictly monotonically until some iterate belongs to an eigenspace associated with λ_1.*

The above result implies that if the computed eigenvalue $\hat{\lambda}$ is accurate enough, *i.e.* it is close enough to one eigenvalue (possibly with multiplicity > 1), then Inverse Iteration converges to the associated eigenspace. In Algorithm 3.4.1, since Inverse Iteration is used if an eigenvalue is returned by bisection in an interval $[\alpha, \beta)$ that is too narrow *i.e.* $|\beta - \alpha| < 2 * \tau$, and the computed eigenvalue is $\hat{\lambda} = (\alpha + \beta)/2$. The eigenvalues λ_i closest to $\hat{\lambda}$ must be in this interval and satisfy

$$|\lambda_i - \hat{\lambda}| < (\beta - \alpha)/2 < \tau,$$

and other eigenvalues λ_j outside of the interval satisfy:

$$|\lambda_j - \hat{\lambda}| > (\beta - \alpha)/2 > |\lambda_i - \hat{\lambda}|.$$

Since τ is the desired accuracy for eigenvalues, $\hat{\lambda}$ is a close enough approximation to the eigenvalues in the interval $[\alpha, \beta)$. So in this case, we take $\hat{\lambda}$ as an approximate eigenvalue to a real eigenvalue with multiplicity $l \geq 1$, or an eigenvalue cluster. The corresponding eigenvectors are computed by combination of Inverse Iteration and Orthogonal Deflation as mentioned in chapter 5.3 to guarantee the orthogonality of eigenvectors within the eigenspace.

6.2.2 Residual and stopping criteria

From

$$r_k = (A - \hat{\lambda}I)x_k = \frac{1}{||z_k||}(A - \hat{\lambda}I)z_k = \frac{1}{||z_k||}x_{k-1}$$

and $||x_{k-1}|| = 1$, it follows:

$$||r_k|| = 1/||z_k||.$$

Therefore the residual is inversely proportional to the norm of the unnormalized iterate and it is used as a criteria for terminating the iterations. Once the residual is small enough, inverse iteration stops because then $(\hat{\lambda}, x_k)$ is an eigenpair of a nearby matrix:

Theorem 6.3 *Let A be a real square matrix and $r_k = (A - \hat{\lambda}I)x_k$ be the residual for some $\hat{\lambda}$ and vector x_k with $||x_k|| = 1$.*

Then there exists a matrix E_k with $(A + E_k - \hat{\lambda}I)x_k = 0$ and $||E_k|| \leq \epsilon$ if and only if $||r_k|| \leq \epsilon$.

Proof 6.1 *Suppose $(A + E_k - \hat{\lambda}I)x_k = 0$ and $||E_k|| \leq \epsilon$. Then*

$$r_k = (A - \hat{\lambda}I)x_k = -E_k x_k$$

implies $||r_k|| \leq ||E_k|| \leq \epsilon$. Now suppose $||r_k|| \leq \epsilon$. Then

$$(A - \hat{\lambda}I)x_k = r_k = r_k x_k^T x_k,$$

implies

$$(A - r_k x_k^T - \hat{\lambda}I)x_k = 0, \quad \textit{and let} E_k = -r_k x_k^T$$

Since E_k is a rank-one matrix,

$$||E_k|| = ||r_k|| \leq \epsilon.$$

Thus, a small residual implies that $\hat{\lambda}$ and x_k are accurate in the backward error sense. In 6.2.3 we show that finite precision has little effect to using $||z_k||$ as a criteria. However, as mentioned in chapter 3.3 and 6.2, an eigenvector with a small backward error is not always orthogonal to other eigenvectors as the orthogonality depends on the gap between eigenvalues. We will revisit this problem and establish orthogonality among eigenvectors in chapter 6.3.

6.2.3 Finite precision residual

It is shown in the previous section that in exact arithmetic, the iterate x_k is an eigenvector of a matrix close to A if its unnormalized version z_k has sufficiently large norm. In fact, it is also true in finite precision arithmetic [32]. In practice, the meaning of "sufficiently large" is determined by the size of the backward error F_k from the linear system solution in the kth iteration.

In the following discussions, we use variables with hats to denote computed values in finite precision arithmetic. *, i.e.* $\hat{x_k}$ stands for the finite precision computed value of x_k, the kth iterate in inverse iteration. The residual of the computed iterate in finite precision:

$$\hat{r}_k \equiv (A - \hat{\lambda}I)\hat{x}_k = -F_k\hat{x}_k + \frac{1}{||\hat{z}_k||}\hat{x}_{k-1}$$

is bounded by

$$\frac{1}{||\hat{z}_k||} - ||F_k|| \leq ||\hat{r}_k|| \leq \frac{1}{||\hat{z}_k||} + ||F_k||.$$

This means, if $\hat{z_k}$ is sufficiently large then the size of the residual is about as small as the backward error. For instance, if

$$||\hat{z}_k|| \geq \frac{1}{c||F_k||}$$

for some $c > 0$, then the residual is at most a multiple of the backward error,

$$||\hat{r}_k|| \leq (1 + c)||F_k||.$$

A lower bound on $||\hat{z_k}||$ can therefore be used as a criterion for terminating the inverse iteration. In our algorithm, we terminate the iteration when

$$||\hat{z}_k||_2 \geq \frac{1}{\tau}.$$

The last concern for Inverse Iteration is that the solution of a ill-conditioned linear system $(A - \hat{\lambda}I)\hat{z}_k = \hat{x}_{k-1}$ would be totally inaccurate when $\hat{\lambda}$ is close to an exact eigenvalue of A. In fact, it was shown by Wilkinson that a computed iterate with a large norm lies in the "correct direction" and "is wrong only by a scalar factor" [26, p342]. Here we illustrate this point by comparing the computed first iterate to the exact first iterate(the same argument applies to any other iterate). The respective exact and finite precision computations are:

$$(A - \hat{\lambda}I)z_1 = x_0, \quad (A - \hat{\lambda}I + F_1)\hat{z}_1 = x_0.$$

For the following theorem we make the standard assumption that $A - \hat{\lambda}I$ is non-singular with respect to the backward error, *i.e.* $||(A - \hat{\lambda}I)^{-1}F_1|| < 1$. The following result assures that the computed iterate is of comparable size to the exact iterate:

Theorem 6.4 *Let $A - \hat{\lambda}I$ be non-singular and $||(A - \hat{\lambda}I)^{-1}F_1|| < 1$. Suppose*

$$(A - \hat{\lambda}I)z_1 = x_0, \quad (A - \hat{\lambda}I + F_1)\hat{z}_1 = x_0.$$

Then

$$||\hat{z}_1|| \geq \frac{1}{2}||z_1||.$$

Proof 6.2 *Since $A - \hat{\lambda}I$ is non-singular*

$$(I + (A - \hat{\lambda}I)^{-1}F_1)\hat{z}_1 = (A - \hat{\lambda}I)^{-1}x_0 = z_1.$$

The assumption $||(A - \hat{\lambda}I)^{-1}F_1|| < 1$ implies that:

$$||z_1|| \leq (1 + ||(A - \hat{\lambda}I)^{-1}F_1||)||\hat{z}_1|| \leq 2||\hat{z}_1||.$$

Since ill-conditioning of the solution does not damage the accuracy of an iterate and we can achieve small backward error F_k with methods discussed in previous sections, *i.e.* Algorithm 4.2 and 5.2.1, it is safe to use the norm of computed iterate $||\hat{z}_k||$ as stopping criterion of Inverse Iteration.

6.3 Orthogonality of eigenvectors

So far we have discussed the backward error of our computed eigenpairs. Let ϵ denote the machine epsilon for a given precision. For Rayleigh Quotient Iteration, if it converges to an eigenpair $(\hat{\lambda}, x)$, it is stopped when:

$$||(A - \hat{\lambda}I)x||_2 < 100n||A||\epsilon.$$

For eigenvalues found by bisection, we use inverse iteration to find the corresponding eigenvectors with stopping criterion

$$||\hat{z}_k|| \geq \frac{1}{n||A||\epsilon},$$

which guarantees a small backward error:

$$||r_k|| \leq \frac{1}{||\hat{z}_k||} + ||F_k|| = O(n||A||\epsilon).$$

The last one of our concerns is the orthogonality of computed eigenvectors, which is related to the backward error, or residuals of the computed eigenpairs. Formally, suppose $\hat{\lambda}_i \neq \hat{\lambda}_j$ are two computed eigenvalues with $\hat{q}_i, \hat{q}_j$ being the computed eigenvectors, then

$$r_i = A\hat{q}_i - \hat{\lambda}_i\hat{q}_i \tag{6.2}$$

and

$$r_j = A\hat{q}_j - \hat{\lambda}_j\hat{q}_j \tag{6.3}$$

Multiplying 6.2 and 6.3 by $\hat{q}_j^T$ and $\hat{q}_i^T$ respectively and taking the difference we have:

$$\hat{q}_i^T\hat{q}_j = (\hat{q}_i^T r_j - \hat{q}_j^T r_i)/(\hat{\lambda}_i - \hat{\lambda}_j)$$

Since the computed eigenvectors both have unit length, taking the norm on both sides of the equation gives:

$$|\hat{q}_i^T\hat{q}_j| \leq (||r_i||_2 + ||r_j||_2)/|\hat{\lambda}_i - \hat{\lambda}_j|.$$

Since we already have small residuals r_i and r_j, the orthogonality of computed eigenvectors depend on the gap of eigenvalues. As discussed in section 3.3, *tol* (e.g 10^{-3}) is used as a threshold value to determine if the absolute gap between eigenvalues is big enough:

1. $\hat{\lambda}_i$ and $\hat{\lambda}_j$ are well separated, $|\hat{\lambda}_i - \hat{\lambda}_j| > ||A|| * tol$;

2. $|\hat{\lambda}_i - \hat{\lambda}_j| < ||A|| * tol$ but they are find in an interval of length smaller than $2||A||n\epsilon$ as a cluster;

3. $|\hat{\lambda}_i - \hat{\lambda}_j| < ||A|| * tol$ and they are not found together in a cluster.

Here we use the parameter $||A|| * tol$ because then if two eigenvalues are well-separated, their eigenvectors will be automatically orthogonal:

$$|\hat{q}_i^T \hat{q}_j| \leq (||r_i||_2 + ||r_j||_2)/|\hat{\lambda}_i - \hat{\lambda}_j| \leq 2 * tol * \epsilon$$

For the second case, our orthogonal deflation guarantees that the eigenvectors within cluster are orthogonal:

$$|\hat{q}_i^T \hat{q}_j| = O(n\epsilon).$$

The last case is detected by keeping a sorted list of computed eigenvalues and check the newly computed eigenvalue against its nearest neighbors. In this case however, we use extended precision to recompute the corresponding eigenvector(s) by Inverse Iteration and Orthogonal Deflation(ref 5.3) if necessary. This will compute the residual to:

$$||r_i|| = O(||A||n\epsilon^2).$$

Therefore even if the eigenvalues are close with respect to ϵ, e.g $|\hat{\lambda}_i - \hat{\lambda}_j| = \Theta(||A||\epsilon)$, we can still get numerically orthogonal eigenvectors:

$$|\hat{q}_i^T \hat{q}_j| = O(n\epsilon).$$

In practice, we use double double arithmetic to mimic quadruple precision arithmetic for efficiency purpose. Our algorithm uses Bailey's double double arithmetic library [28] and QBLAS packages [60] for some BLAS implementations. Although it is expensive(about 20x slower) to perform extended precision arithmetic, it is rarely needed.

6.4 Backward error

Let $\hat{\Lambda}$ denote the matrix of computed real eigenvalues and $\hat{V}$ denote the matrix of computed real eigenvectors. In order to assess the accuracy of the computed eigenvalues $\hat{\Lambda}$ and eigenvectors $\hat{V}$ in the backward sense, we would like to show that $\hat{\Lambda}$ and $\hat{V}$ represent an exact eigenvalue decomposition of some symmetric matrix. There are two obvious backward errors $\hat{E}$ and E by $A + \hat{E} = \hat{V}\hat{\Lambda}\hat{V}^T$ and $A + E = \hat{V}\hat{\Lambda}\hat{V}^{-1}$. The relation between the two backward errors are examined in [14] and we adopt the first version here.

Let

$$\epsilon_r \equiv ||A\hat{V} - \hat{V}\hat{\Lambda}||, \quad \epsilon_o \equiv ||I - \hat{V}^T\hat{V}|| = ||I - \hat{V}\hat{V}^T||$$

denote the residual of eigenpairs and the deviation from orthogonality respectively and define a backward error $\hat{E}$ by:

$$A + \hat{E} = \hat{V}\hat{\Lambda}\hat{V}^T.$$

Then $\hat{E}$ is small if both residual and deviation from orthogonality are small, i.e:

$$\begin{aligned}
||\hat{E}||_2 &= ||\hat{V}\hat{\Lambda}\hat{V}^T - A||_2 \\
&= ||(\hat{V}\hat{\Lambda}\hat{V}^T - A\hat{V}\hat{V}^T) + (A\hat{V}\hat{V}^T - A)||_2 \\
&= ||(\hat{V}\hat{\Lambda} - A\hat{V})\hat{V}^T + A(\hat{V}\hat{V}^T - I)||_2 \\
&\leq \epsilon_r||\hat{V}^T||_2 + \epsilon_o||A||_2
\end{aligned}$$

Previously we have examined the residual ϵ_r and deviation from orthogonality ϵ_o and we have:

$$\epsilon_r = O(n\epsilon||A||_2), \quad \epsilon_o = O(n\epsilon).$$

Hence, the backward error $\hat{E}$ is also small:

$$||E_2|| = O(n\epsilon||A||_2).$$

Chapter 7

Numerical Experiments and conclusions

In this section we present several experimental results to illustrate the performance of our proposed eigensolver (Algorithm 3.4.1). Instead of comparing with other eigensolvers, our results focus on two aspects: **Accuracy** and **Complexity**.We would like to illustrate by numerical experiments that the proposed algorithm (Algorithm 3.4.1) has achieved the following goals:

1. Efficiency. We present run-time performance showing that the asymptotic complexity of Algorithm 3.4.1 is $O(n^2r^2)$ when r is kept as a constant;

2. Accuracy. We illustrate by residual norm that the computed eigensystem is accurate relative to the norm of input matrix A;

3. Orthogonality. The computed eigenvectors are numerically orthogonal for matrices with separated spectrum and those with eigenvalue clusters;

4. Computational efforts. In addition, we also include stats such as "Average number of RQI and inverse iteration for each eigenpair" and "total number of QR decomposition" to illustrate that our iterative methods do converge quickly as proposed and the less efficient backup method is used very rarely.

The quality of the computed eigensystem is measured by two parameters, γ and η defined above. Let $\hat{\Lambda}, \hat{V}$ denote the computed eigenvalues and eigenvectors of A. Then γ represents the size of the residual:

$$\gamma \equiv \frac{||A\hat{V} - \hat{V}\hat{\Lambda}||}{n||A||_2}.$$

The division by $n||A||_2$ is aimed to determine the results relatively measured against the "roundoff" error.

The next parameter η measures the deviation from orthogonality:

$$\eta = \frac{||I - \hat{V}^T\hat{V}||}{n}.$$

Again the division by n is aimed to determine the relative deviation against the "roundoff" error. The following table shows the metrics for matrices with increasing sizes.

Matrix Type	Matrix size n	Avg number of RQI and Inverse Iter	Total number of QR	Residual γ	Orthogonality η
No Cluster	1000	5.19	6	$1.57E-17$	$2.4E-16$
No Cluster	2000	5.45	8	$1.01E-17$	$2.3E-16$
No Cluster	3000	5.48	12	$2.54E-17$	$2.2E-16$
No Cluster	4000	5.50	10	$1.46E-16$	$4.4E-16$
No Cluster	5000	5.56	15	$0.91E-16$	$6.2E-14$
No Cluster	6000	5.49	11	$3.16E-16$	$5.3E-14$
No Cluster	7000	5.53	18	$1.18E-17$	$2.1E-14$
No Cluster	8000	5.52	17	$8.08E-16$	$3.4E-14$
No Cluster	9000	5.60	20	$1.01E-17$	$2.2E-14$
No Cluster	10000	5.54	22	$0.42E-16$	$1.3E-14$
Clustered	1000	5.26	20	$1.41E-17$	$3.3E-16$
Clustered	2000	5.49	25	$1.23E-17$	$2.2E-16$
Clustered	4000	6.50	48	$9.85E-16$	$2.5E-16$
Clustered	8000	6.37	70	$4.00E-16$	$5.5E-15$

In the above table we display the test results of Algorithm 3.4.1 on two types of matrices: synthetic matrices with well-separated spectrum, and matrices with random clusters and cluster sizes. It can be seen from the table that Algorithm 3.4.1 takes almost a consistent number of Rayleigh Quotient Iteration and inverse iterations for each eigenpair. This is credited to the cubic local convergence property of Rayleigh Quotient Iteration. After several eigenpairs have been found by RQI, it may converge to an already found eigenpair, or converge outside of all the intervals. In those situations, bisection was invoked until an interval gets too narrow. The corresponding eigenvectors are then computed by inverse iterations. For matrices with eigenvalue clusters, the average number of RQI and inverse iterations go up due to the orthogonal deflation algorithm (Algorithm 5.3.1) used to re-compute the eigenvectors corresponding to eigenvalue clusters. In addition, eigenvalue clusters have also caused the LDL^T

decomposition algorithm(Algorithm 4.2) to become more unstable, which is reflected as the increase in the number of QR decomposition for matrices with eigenvalue clusters.

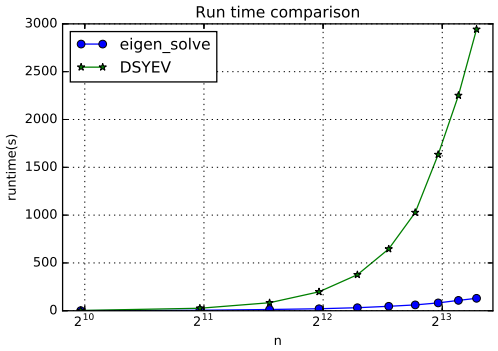

Figure 7.1: Run time performance for unclustered matrices

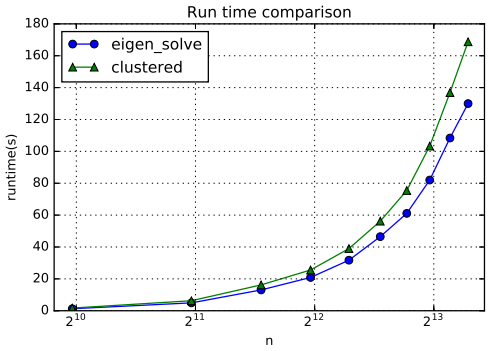

Figure 7.2: Clustered matrices v. s Unclustered matrices

The first plot above illustrates the run time performance of our algorithm against the naive method, *i.e.* forming $A = D + UHU^T$ then use a usual eigensolver. This is to show that our algorithm has $O(n^2)$ asymptotic complexity. The second plot shows that the orthogonal deflation and extended precision used for eigenvalue clusters add some overhead to the run time but still keep the algorithm quadratic.

7.1 Concluding Remarks

In this thesis, we have studied the eigenproblem of a special type of matrix, *i.e.* a diagonal matrix with symmetric low rank perturbation. We propose an efficient method to compute accurate eigenvalues as well as numerically orthogonal eigenvectors. In addition, we make use of a combination of techniques to solve the loss of orthogonality issue faced by the eigenproblem and as a result, our algorithm (Algorithm 3.4.1) does not require any sort of re-orthogonalization routines such as the Gram-Schmidt process. Our proposed algorithm is based on bracketing algorithm, which finds eigenpairs separately one at a time. This property exhibits potentials for parallel computing. However, the unpredictability of Rayleigh Quotient Iteration prevents our algorithm from delivering a subset of eigenpairs at a reduced cost.

Now we would like to point out some of the key guiding principals that drive the development of our proposed algorithm. We think that they are not only limited to this problem and can be applied to other numerical linear algebra procedures.

- **Try to optimize for the main part of the computation.** In our proposed algorithm, Rayleigh Quotient Iteration accounts for almost all computational costs. One primary requirement for it to be efficient is an efficient method to solve

the shifted linear system. When developing the method, we were well aware that efficient decomposition based methods do not always guarantee stability, but the backward error of the solution is crucial to the accuracy of the computed eigenpair (ref chapter 6.2.3). However, since the numerical stability issue arises not so frequently, we adopt the approach to use a fast but unstable method as our main decomposition method (algorithm 4.2) and back it up with a slower but stable method(algorithm 5.2.1) to handle numerical issues. In this way, we are able to achieve high efficiency in most of our computations and still guarantee numerical stability.

- **Find appropriate transformation of problems.** When finding a stable method for solving the shifted linear system, QR decomposition was an obvious choice. Although counter intuitive at first, it turned out that transformation the linear system into a larger one actually enabled the use of Householder QR decomposition. (see chapter 5.1). For the computation of eigenvectors corresponding to eigenvalue clusters, we faced a very similar problem as the "deflation" method used in Power Iteration. We adopted the idea of "deflation" but combined it with orthogonal transformation to preserve the eigenvalues and guarantee orthogonality of eigenvectors.

As mentioned in chapter 2.2.2, our eigenproblem $D+UHU^T$ appears in the synthesis step of the divide and conquer algorithm for symmetric band matrix. One important future development of our proposed algorithm is to be incorporated as a solver for the synthesis step and enable direct divide and conquer method on symmetric band matrix.

Another direction is to explore parallel opportunities for the algorithm. This can come both as a way to speed up the bracketing algorithm with threads and making it viable to matrices of larger sizes.

References

[1] A.M.Ostrowski. "On the convergence of Rayleigh quotient iteration for the computation of characteristic roots and vectors, I-VI". In: *Arch. Rational. Mech. Anal* 1-4 (1958).

[2] E. Anderson et al. *LAPACK Users' Guide (Third Ed.)* Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1999.

[3] Michael Anderson et al. "Communication-Avoiding QR Decomposition for GPUs". In: *Proceedings of the 2011 IEEE International Parallel & Distributed Processing Symposium*. IPDPS '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 48–58.

[4] Peter Arbenz. "Divide and conquer algorithms for the band symmetric eigenvalue problem". In: *Parallel Computing* 8 (1992), pp. 1105–1128.

[5] Peter Arbenz and G.H.Golub. "On the spectral decomposition of Hermitian matrices modified by indefinite low rank perturbations and its applications". In: *SIAM, Journey Matrix Analysis and Applications* 9 (1988), pp. 40–58.

[6] Grey Ballard, James Demmel, and Nicholas Knight. "Avoiding Communication in Successive Band Reduction". In: *ACM Trans. Parallel Comput.* 1.2 (2015), 11:1–11:37.

[7] C. H. Bischof and C. V. Loan. *The WY Representation for Products of Householder Matrices*. Cornell University, Ithaca, NY, USA, 1985.

[8] B.N.Partlett and W.Kahan. "On the convergence of a practical QR algorithm. (with discussion)". In: *Information Processing 68* I (1969).

[9] Jan H. Brandts and Ricardo Reis da Silva. "Computable eigenvalue bounds for rank-k perturbations". In: *Linear Algebra and its Applications* 432.12 (2010), pp. 3100–3116.

[10] Andreas Buja et al. "Data Visualization With Multidimensional Scaling". In: *Journal of Computational and Graphical Statistics* 17.2 (2008), pp. 444–472.

[11] James R. Bunch and Linda Kaufman. "Some Stable Methods for Calculating Inertia and Solving Symmetric Linear Systems". In: *Mathematics of Computation* 31 (1977), pp. 163–179.

[12] James R. Bunch, Christopher P. Nielsen, and Danny C. Sorensen. "Rank-one Modification of the Symmetric Eigenproblem". In: *Numerische Mathematik* 31.1 (1978), pp. 31–48.

[13] Cheryl M. M. Carey et al. *A New Approach for Solving Perturbed Symmetric Eigenvalue Problems*. Stanford, CA, USA, 1992.

[14] S. Chandrasekaran and I.C.F. Ipsen. "Backward Errors for Eigenvalue and Singular Value Decompositions". In: *Numerische Mathematik* 68 (1994), pp. 215–223.

[15] Cholesky Decomposition. *Cholesky Decomposition — Wikipedia, The Free Encyclopedia*. 2018. URL: `https://en.wikipedia.org/wiki/Cholesky_decomposition`.

[16] J. J. Cuppen. "A Divide and Conquer Method for the Symmetric Tridiagonal Eigenproblem". In: *Numerische Mathematik* 36.2 (June 1980), pp. 177–195.

[17] Achiya Dax. "The Orthogonal Rayleigh Quotient Iteration (ORQI) method". In: *Linear Algebra and its Applications* 358.1 (2003), pp. 23–43.

[18] James W. Demmel. *Applied Numerical Linear Algebra*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1997.

[19] James W Demmel, Dhillon Inderjit, and Huan Ren. "On the correctness of some bisection-like parallel eigenvalue algorithms in floating point arithmetic." In: *ETNA. Electronic Transactions on Numerical Analysis [electronic only]* 3 (1995), pp. 116–149.

[20] Inderjit Singh Dhillon. *A New $O(n^2)$ Algorithm for the Symmetric Tridiagonal Eigenvalue/Eigenvector Problem*. EECS Department, University of California, Berkeley, Oct. 1997.

[21] Jiu Ding and Aihui Zhou. "Eigenvalues of rank-one updated matrices with some applications". In: *Applied Mathematics Letters* 20 (2007), pp. 1223–1226.

[22] Wilfried N. Gansterer, Robert C. Ward, and Richard Muller. "An Extension of the Divide-and-conquer Method for a Class of Symmetric Block-tridiagonal Eigenproblems". In: *ACM Trans. Math. Softw.* 28.1 (2002), pp. 45–58.

[23] Wilfried N. Gansterer et al. *Computing Approximate Eigenpairs of Symmetric Block Tridiagonal Matrices*. 2003.

[24] Wallace Givens. "The Characteristic Value-Vector Problem". In: *Journal of the ACM* 4.3 (1957), pp. 298–307.

[25] Gene H. Golub and Charles F. Van Loan. *Matrix Computations (3rd Ed.)* Baltimore, MD, USA: Johns Hopkins University Press, 1996.

[26] G.Peters and J.Wilkinson. "Inverse iteration, ill-conditioned equations and Newton's method". In: *SIAM Review* 21 (1979), pp. 339–360.

[27] Azzam Haidar, Hatem Ltaief, and Jack Dongarra. "Toward a High Performance Tile Divide and Conquer Algorithm for the Dense Symmetric Eigenvalue Problem". In: *SIAM Journal on Scientific Computing* 34 (Feb. 2012), pp. 249–274.

[28] Yozo Hida, Sherry Li, and David Bailey. *Library for Double-Double and Quad-Double Arithmetic*. Jan. 2008.

[29] Nicholas J. Higham and Desmond J. Higham. "Large Growth Factors in Gaussian Elimination with pivoting". In: *SIAM Journal on Matrix Analysis and Applications* 10.2 (Sept. 1989), pp. 155–164.

[30] Alston S. Householder. "Unitary Triangularization of a Nonsymmetric Matrix". In: *Journal of the ACM* 5.4 (Oct. 1958), pp. 339–342.

[31] H.Wielandt. *Beitrage zur mathematischen Behandlung komplexer Eigenwertprobleme, Teil V: Bestimmung hoherer Eigenwerte durch gebrochene Iteration*. Bericht B 44/J/37. 1944.

[32] Ilse C. F. Ipsen. "Computing an Eigenvector with Inverse Iteration". In: *SIAM Review* 39.2 (1997), pp. 254–291.

[33] Andrew Kerr, Dan Campbell, and Mark Richards. "QR Decomposition on GPUs". In: *Proceedings of 2Nd Workshop on General Purpose Processing on Graphics Processing Units*. GPGPU-2. Washington, D.C., USA: ACM, 2009, pp. 71–78.

[34] Jim Lambers. *Summer Session 2009-2010, section 4 notes*. 2009.

[35] P. Luszczek, H. Ltaief, and J. Dongarra. "Two-Stage Tridiagonal Reduction for Dense Symmetric Matrices Using Tile Algorithms on Multicore Architectures". In: *2011 IEEE International Parallel Distributed Processing Symposium*. 2011, pp. 944–955.

[36] Roy Mitz, Nir Sharon, and Yoel Shkolnisky. *Symmetric rank one updating from partial spectrum with an application to out-of-sample extension*. 2017. eprint: `arXiv:1710.02774`.

[37] Michael Moldaschl and Wilfried N. Gansterer. "Comparison of Eigensolvers for Symmetric Band Matrices". In: *Sci. Comput. Program.* 90.PA (2014), pp. 55–66. ISSN: 0167-6423.

[38] N.J.Higham. "Stability of diagonal pivoting method with partial pivot". In: *SIAM Journal on Matrix Analysis and Application* 18 (1997), pp. 52–65.

[39] HyungSeon Oh and Zhe Hu. "Multiple-rank modification of symmetric eigenvalue problem". In: *MethodsX* 5 (2018), pp. 103–117.

[40] J.M Ortega. *Numerical analysis, a second course.* New York: Academic Press, 1972.

[41] Lawrence Page et al. *The PageRank Citation Ranking: Bringing Order to the Web.* 1999-66. Stanford InfoLab, 1999.

[42] B. N. Parlett. "Laguerre's Method Applied to the Matrix Eigenvalue Problem". In: *Mathematics of Computation* 18 (1964), pp. 464–485.

[43] B. N. Parlett. "The Rayleigh Quotient Iteration and Some Generalizations for Nonnormal Matrices". In: *Mathematics of Computation* 28 (1974), pp. 679–693.

[44] Beresford N. Parlett. *The Symmetric Eigenvalue Problem*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1998.

[45] Walter Gander Peter Arbenz and Gene H.Golub. "Restricted rank modification of the symmetric eigenvalue problem: Theoretical considerations". In: *Linear Algebra and its Applications* (1988), pp. 75–95.

[46] G. Peters and J.H. Wilkinson. "Inverse iteration, ill-conditioned equations and Newton's method". In: *SIAM Review* 21 (1979), pp. 339–360.

[47] Sam T. Roweis and Lawrence K. Saul. "Nonlinear dimensionality reduction by locally linear embedding". In: *SCIENCE* 290 (2000), pp. 2323–2326.

[48] R.P.Brent. *Algorithms for minimization without derivatives*. Prentice-Hall, 1973.

[49] Jeffery D. Rutter. *A Serial Implementation of Cuppen's Divide and Conquer Algorithm for the Symmetric Eigenvalue Problem*. EECS Department, University of California, Berkeley, 1994.

[50] Yousef Saad. *Numerical Methods for Large Eigenvalue Problems (2nd Ed.)* Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2011.

[51] Inderjit S.Dhillon and Bersford N. Parlett. "Orthogonal Eigenvectors and Relative Gaps". In: *SIAM Journal on Matrix Analysis and Application* 25.3 (2003), pp. 858–899.

[52] Jonathon Shlens. *A Tutorial on Principal Component Analysis*. 2014. eprint: `arXiv:1404.1100`.

[53] B. T. Smith et al. *Matrix Eigensystem Routines - EISPACK Guide, volume 6 of Lecture Notes in Computer Science*. Berlin: Springer-Verlag, 1976.

[54] R.C. Thompson. "The behavior of eigenvalues and singular values under perturbations of restricted rank". In: *Linear Algebra and its Applications* 13.1 (1976), pp. 69–78.

[55] J. M. Varah. "Rigorous Machine Bounds for the Eigensystem of a General Complex Matrix." In: *Mathematics of Computation* 22.104 (1968), pp. 793–801.

[56] W.Hakan. *Notes on Laguerre's iteration*. University of California Computer Science Division preprint, 1992.

[57] J. H. Wilkinson. "Rigorous Error Bounds for Computer Eigensystems". In: *The Computer Journal* 4.3 (1961), pp. 230–241.

[58] James H. Wilkinson. *Rounding Errors in Algebraic Processes*. New York, NY, USA: Dover Publications, Inc., 1994.

[59] C.W.Ueberhuber W.N.Gansterer J.Schneid. "A low-complexity divide and conquer method for computing the eigenvalues and eigenvectors of symmetric band matrices". In: *BIT. Numerical mathematics* 41 (2001), pp. 967–976.

[60] S. Yamada et al. "Quadruple-precision BLAS using Bailey's arithmetic with FMA instruction: its performance and applications". In: *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 2017, pp. 1418–1425.

[61] Y.Saad. *Deflation*. 2000. URL: `http://www.netlib.org/utk/people/JackDongarra/etemplates/node219.html`.