Accelerating the Arnoldi Iteration - Theory and Practice

Chao Yang

CRPC-TR98748-S
April 1998

Center for Research on Parallel Computation
Rice University
6100 South Main Street
CRPC - MS 41
Houston, TX 77005

Submitted August 1998

RICE UNIVERSITY

Accelerating the Arnoldi Iteration – Theory and Practice

by

Chao Yang

A Thesis Submitted
in Partial Fulfillment of the
Requirements for the Degree

Doctor of Philosophy

Approved, Thesis Committee:

Danny C. Sorensen, Chairman
Professor of Computational and Applied Mathematics

Steven J. Cox
Associate Professor of Computational and Applied Mathematics

Matthias Heinkenschloss
Associate Professor of Computational and Applied Mathematics

John C. Polking
Professor of Mathematics

Houston, Texas

May, 1998

Abstract

Accelerating the Arnoldi Iteration – Theory and Practice

by

Chao Yang

The Arnoldi iteration is widely used to compute a few eigenvalues of a large sparse or structured matrix. However, the method may suffer from slow convergence when the desired eigenvalues are not dominant or well separated. A systematic approach is taken in this dissertation to address the issue of how to accelerate the convergence of the Arnoldi algorithm within a subspace of limited size.

The acceleration strategies presented here are grouped into three categories. They are the method of restarting, the method of spectral transformation and the Newton-like acceleration.

Simply put, the method of restarting repeats a k-step Arnoldi iteration after improving the starting vector. The method is further divided into polynomial and rational restarting based on the way the starting vector is modified. We show that both mechanisms can be implemented in an implicit fashion by relating the restarted Arnoldi to a truncated QR or the RQ iteration. The rational restarting via a Truncated RQ (TRQ) iteration converges extremely fast. However, a linear system must be solved for each restarting. The possibility of replacing a direct linear

solver with a preconditioned iterative solver while maintaining the rapid convergence of TRQ is explored in this thesis.

The method of spectral transformation is based on the idea of transforming the original eigenvalue problem into one that is easier to solve. Again, both polynomial and rational transformations are possible. Practical issues regarding the design and implementation of effective spectral transformations are discussed.

Finally, one can treat the eigenvalue problem as a nonlinear equation upon which Newton-like methods can be applied. The Jacobi-Davidson (JD) algorithm proposed by Sleijpen and Van der Vorst takes this approach. In JD, the Arnoldi iteration is merely used as a global searching tool to provide a good starting point for the Newton iteration. This algorithm shares many similar properties with the TRQ iteration. Numerical comparisons between these two methods are made in this thesis.

Acknowledgments

I am deeply indebted to my thesis advisor, Professor Dan Sorensen, for his support and friendship. Dan has created many opportunities for me to work on interesting and important problems in scientific computing. He set a high standard for me and taught me to be resilient in solving difficult research problems.

I would also like to thank other members of my thesis committee for their interest in my thesis. I thank Rich Lehoucq for introducing me to the ARPACK software project which led me into this fascinating area of large scale eigenvalue computation. I benefited from many discussions with him. I am grateful to Ralph Byers for giving me good advice and encouraging me to continue my graduate study at Rice. Special thanks go to John Lewis at the Boeing Company for inviting me to work there on some very interesting aspects of the eigenvalue problems. I would also like to thank Ed Rothberg, Giacomo Brussino and Mimi Celis at Silicon Graphics, Inc. for giving me the opportunity to work on some industrial eigenvalue problems. I wish to thank Kristi Maschhoff for her help, encouragement and friendship. I would also like to thank Keith Berrier and Denise Huang for reading my thesis thoroughly and pointing out many typos and mistakes.

The Department of Computational and Applied Mathematics (CAAM) has provided a stimulating environment and friendly atmosphere during my graduate study. I would like to thank all CAAM students, especially my officemates Jen Rich and Erica Zimmer, for their friendship and help.

Finally, I would like to thank my parents Chuanhou Yang and Yuhua Zhao for their encouragement and support. Many thanks are due to my host family Roy

and Sue Stubbs in Warrensburg, Missouri for their support ever since I came to the States.

Contents

Abstract ii

Acknowledgments iv

List of Illustrations ix

List of Tables xiv

1 Introduction 1

1.1 The Eigenvalue Problem 2

1.2 The Power Method 4

1.3 The Arnoldi Method 6

1.4 Organization of the Thesis 8

1.5 Notation 9

2 Polynomial Restarting and IRA 11

2.1 The Implicitly Shifted QR Algorithm 14

2.2 Implicitly Restarted Arnoldi 16

2.3 Choice of shifts 17

2.4 Numerical Examples 20

2.4.1 One-dimensional Laplacian 21

2.4.2 Crystal Growth 26

3 Rational Restarting and Truncated RQ-iteration 33

3.1 Truncating an RQ iteration 34

3.2 Implementation issues in TRQ 42

3.2.1 Solving the TRQ Equations 42

3.2.2 Selection of Shifts 46
3.2.3 Deflation 49
3.3 Numerical Examples 50
3.4 Inexact TRQ 51
3.4.1 The Algorithm 52
3.4.2 Convergence Analysis 56
3.4.3 Numerical Examples 63

4 Spectral Transformation 71

4.1 A Practical Use of the Shifted and Inverted Lanczos 72
4.1.1 The Boeing Heuristics 75
4.1.2 Adding Heuristics to IRL 77
4.1.3 How to Choose the Target Shift 81
4.1.4 Numerical Experiment 85
4.2 The Rational Krylov Method 89

5 Polynomial Spectral Transformation 92

5.1 Chebyshev and Kernel polynomials 93
5.2 Leja Points and Related Polynomials 98
5.3 Minmax Approximation and the Remez Algorithm 100
5.4 Minmax Polynomial Acceleration 104
5.4.1 Minmax Approximation to $\frac{1}{\lambda}$ 104
5.4.2 Near Minmax Approximation 105
5.4.3 Constrained Minmax Approximation 108
5.5 Least Square Approximation 109
5.6 Polynomial Accelerants for Computing Interior Eigenvalues 114

5.6.1 Chebyshev and Kernel Polynomials 115
5.6.2 Minmax Polynomials 118
5.6.3 Least Square Polynomials 119
5.7 Numerical Examples 119

6 Newton-like Acceleration 128
6.1 Basic Formulation 129
6.1.1 Eigenvector Correction 129
6.1.2 Subspace Augmentation 132
6.1.3 Inexact Newton Correction 134
6.1.4 Termination and Restart 135
6.1.5 Generalized Eigenvalue Problems 135
6.2 JDQR and JDQZ 136
6.3 Solving the Correction Equation 143
6.4 Comparison with ITRQ 148

7 Thesis Summary and Future Work 150

Bibliography 155

Appendix 170

Illustrations

1.1 The power iteration. 4

1.2 The Arnoldi iteration. 7

2.1 One cycle of the Implicitly Restarted Arnoldi Iteration. The shaded area contains nonzero entries. The Arnoldi factorization (a) is modified by a QR update (b) which produces a new factorization of length k (c). The new factorization is extended to the last column of V_m and H_m by running p additional Arnoldi iterations (d). 18

2.2 The filtering polynomial produced by IRL with exact shifts. The circles denote $p(\lambda_j)$, where λ_j $(j = 2, 2, ..., 100)$ are eigenvalues of A. . 24

2.3 Distribution of the roots of the cumulative filtering polynomials at each IRL sweep. The top plot was generated from the exact shifts applied during the IRL iteration. The bottom plot corresponds to IRL with Leja points. 26

2.4 The sparsity pattern of A. 30

2.5 The spectrum of A. 31

2.6 A 30-th degree filtering polynomial generated just before IRA converged. 32

2.7 A different view of the filtering polynomial generated just before IRA converged. 32

3.1 Implicitly Shifted RQ-iteration. 35

3.2 Truncating the RQ iteration. 37
3.3 The Truncated *RQ*-iteration. 42
3.4 Direct Solution of the TRQ Equations. 46
3.5 Inexact TRQ iteration. 56
3.6 The 32 rightmost eigenvalues of a 200×200 BWM matrix. 66
3.7 The convergence history of ITRQ. 66
3.8 Schur-Wielandt Deflated Inverse Iteration. 68
3.9 The sparsity pattern of the reactive scattering matrix. 69
3.10 Comparison of (preconditioned) ITRQ with IRA. 70

4.1 A flow chart for concatenating several Lanczos runs together. 73
4.2 Effect of spectral transformation $\psi(\lambda) = \frac{1}{\lambda - \sigma}$. 74
4.3 Split the spectrum of interest into several subsets and combine several Lanczos runs with different target shifts to capture all desired eigenvalues. 75
4.4 The location of the next target shift. 82
4.5 Another way of choosing the next target shift. 82
4.6 Ritz values interlace with converged eigenvalues. 83
4.7 Place the target shift between the last two converged eigenvalues to avoid missing clustered eigenvalues. 84
4.8 Increasing the dimension of the Krylov subspace $(k + p)$ lowers the overall computational cost of the multi-shift Lanczos driver. 86
4.9 Rational Krylov Sequence Iteration. 90

5.1 The solid curve corresponds to a 10-th degree Chebyshev polynomial with $\alpha = 0.02$ and $\beta = 1$. The dash-dot curve corresponds to a 10-th degree polynomial $K(\lambda; 0)$ constructed using Chebyshev polynomials with $\alpha = 0.01$ and $\beta = 1$. 95

5.2 Chebyshev polynomials $C_{10}(\lambda; \alpha, \beta)$, where $\beta = 1.0$ and α is determined from (5.1) for various values of Δ. The vertical dash-dotted line indicates the location of α. 96

5.3 Kernel and Chebyshev polynomials on $[0.001, 1]$ with different α's. Solid curves correspond to Chebyshev polynomials and dash-dotted curves correspond to Kernel polynomials. All Chebyshev polynomials are scaled so that they have the same magnitude as the corresponding Kernel polynomial at $\lambda = 0.001$. 98

5.4 Polynomials constructed with Leja points. The solid curve corresponds to a polynomial whose roots are placed at Leja points associated with the weight function $w(\lambda) = |\lambda - 0.001|$. The dotted curve is produced by interpolating $\frac{1}{\lambda}$ at the Leja points associated with $w(\lambda) = 1$. The dashed curve represents a polynomial whose roots are the Leja points of $[0.1, 1]$ associated with the weight function $w(\lambda) = 1.0$. 100

5.5 Remez Exchange Algorithm for constructing a minmax polynomial approximant to $f(\lambda)$. 103

5.6 Minmax polynomial approximation to $\psi(\lambda) = 1/\lambda$ on the interval $[10^{-3}, 1]$. The dash-dotted curve corresponds to a 10-th degree polynomial constructed by the Remez algorithm. The solid curve corresponds to a 50-th degree polynomial. The dotted curve is the function $\psi(\lambda) = 1/\lambda$. 105

5.7 Approximation $1/\lambda$ by 50-th degree minmax and near minmax polynomials on $[10^{-3}, 1]$. 107

5.8 Constrained and unconstrained minmax polynomial approximation to $\psi(\lambda) = \frac{1}{\lambda}$ on the interval $[10^{-3}, 1]$. The solid curve corresponds to a 50-th degree constrained minmax polynomial. The dashed curve corresponds to a 50-th degree unconstrained minmax polynomial. The dotted curve is the function $\psi(\lambda) = \frac{1}{\lambda}$. 110

5.9 The solid curve corresponds to a 10-th degree Chebyshev least squares polynomial approximation to $\frac{1}{\lambda}$ constructed on $[0.001, 1]$. The dash-dot curve corresponds to a Legend least squares polynomial approximant of the same degree. 112

5.10 The solid curve corresponds to a 10-th degree Ritz least squares polynomial implicitly constructed by applying a 10-step Lanczos run to A. The transformed eigenvalues are plotted in circles. 115

5.11 The solid curve shown in the top figure corresponds to a 50-th degree Ritz least squares polynomial. The circles in that figure indicate the location of the transformed eigenvalues. The dotted curve corresponds to the rational function $\frac{1}{\lambda}$. The solid curve in the bottom figure shows the absolute error $e(\lambda) = \frac{1}{\lambda} - p_m(\lambda)$ of the approximation. 116

5.12 The solid curve corresponds to a 20-th degree Chebyshev polynomial constructed with $\alpha = 0.01$ and $\beta = 1$. The dash-dotted curve corresponds to a Kernel polynomial $K(\lambda; 0)$ constructed using with a Chebyshev basis $\{T_j(\lambda; -1, 1)\}$. 117

5.13 Symmetric and Non-symmetric Bandpass polynomials. The polynomials shown in the left graph are define on an interval that is symmetric about zero. The polynomial generated by a direct Remez procedure gives a narrow passband bandwidth on a non-symmetric interval. 120

5.14 Comparison of various polynomial transformations applied to the reactive scattering eigenvalue calculation. 123

5.15 The sparsity pattern of the Anderson matrix. 124

5.16 Comparison of various polynomial transformations applied to the Anderson model eigenvalue calculation. 125

6.1 The Jacobi-Davidson Iteration. 133

6.2 The Jacobi-Davidson QR Iteration 139

6.3 The convergence history of JDQR and ITRQ for the CK656 matrix. . 149

7.1 The spectrum of a discretized damped string operator 154

7.2 Doubly shifted TRQ iteration 171

Tables

2.1 Ritz values computed from a 90-step standard Lanczos run with reorthogonalization. 22

2.2 Ritz values computed from a 120-step standard Lanczos run with no reorthogonalization. 23

2.3 Comparison of three shifting strategies 23

2.4 Performance of Chebyshev shifts with different α values. 25

2.5 Comparison of matrix-vector multiplications for three shifting strategies with small p values. 25

3.1 Convergence history of the 4 computed eigenvalues of a 2-D Laplacian. 52

3.2 The convergence of inexact TRQ. 65

4.1 Components of the cost function associated with a k-step Lanczos run. 76

4.2 Comparison of the single-shift ARPACK run with the multiple-shift variant. 86

4.3 Detailed view of the multi-shift Lanczos process 87

4.4 Percentage of CPU time spent in various parts of the computation. . 88

5.1 Comparison of polynomial transformations 121

5.2 Comparison of bandpass polynomial transformations 122

5.3 Computed eigenvalues of the Anderson model 126

Chapter 1

Introduction

The Arnoldi method [2] is widely used for computing eigenvalues and eigenvectors of large sparse and/or structured matrices. This thesis explores various techniques for improving the convergence of the Arnoldi method. The purpose of this chapter is to provide some background on large-scale matrix eigenvalue problems as well as the Arnoldi method.

We begin by pointing out the main sources of large-scale eigenvalue problems and the context in which an iterative eigenvalue solver is of enormous value. Although the main theme of this dissertation is how to accelerate the Arnoldi method, we also wish to emphasize the importance of a much simpler, yet useful algorithm – the *power method*. The power method is the driving force for many of the acceleration schemes to be discussed in this thesis. Despite its primitive nature, it has been a building block for developing and analyzing modern iterative methods for solving eigenvalue problems. We will examine the many facets of this method in Section 1.2.

The Arnoldi method is defined in Section 1.3 along with some standard terminologies. In that section, we will also briefly describe the convergence pattern of the eigenvalue approximation and thus give motivation to the need for developing acceleration strategies.

Finally, we outline the organization of the remainder of the thesis in Section 1.4 and establish some convention for notation in Section 1.5.

1.1 The Eigenvalue Problem

The algebraic eigenvalue problem

$$Ax = \lambda x \tag{1.1}$$

is one of the fundamental problems in linear algebra. This problem can be found in many disciplines such as mechanics, system theory, ecology and economics [10, pp. 111-147] [75, pp. 303-321]. The traditional method for solving (1.1) is the QR algorithm [22, 23]. This algorithm applies a sequence of unitary similarity transformations to $A \in \mathbb{C}^{n\times n}$ to turn it into an equivalent triangular matrix R with eigenvalues exposed on the diagonal. The number of floating point operations (FLOPS) required by the QR algorithm is proportional to n^3. The storage required is proportional to n^2. When n becomes large, the algorithm becomes expensive to use.

Many large-scale eigenvalue problems arise from:

1. Finite dimensional approximations to a continuous model

$$\mathcal{L}f = \lambda f, \tag{1.2}$$

   where $\mathcal{L}$ is a linear differential or integral operator and f is a function that belongs to an appropriate space.

2. The study of a linearly coupled system characterized by the differential equation

$$\frac{du}{dt} = Au + f, \tag{1.3}$$

   where $u \in \mathbb{C}^{n\times 1}$ represents the states of various components within the system, $A \in \mathbb{C}^{n\times n}$ describes how they are related and $f \in \mathbb{C}^{n\times 1}$ is usually an external input.

These problems are often sparse or structured. By that, we mean the matrix A has very few nonzero elements, or it has a special structure that allows $y \leftarrow Ax$ to be implemented efficiently in much less than n^2 FLOPS. (An example of this is the discrete Fourier transform matrix.) Typically, only a small subset of the spectrum is of interest. Thus an algorithm like QR will not only be expensive but also wasteful.

Since the 1970's, Krylov subspace methods such as the Arnoldi iteration have become popular for solving large-scale eigenvalue problems. These methods seek the solution to (1.1) from a *Krylov subspace*:

$$\mathcal{K}(A, v_0; k) = \{v_0, Av_0, ..., A^{k-1}v_0\}.$$

No unitary transformation is applied. One only needs to provide a matrix vector multiplication routine to carry out the iteration. The storage requirement of the method is on the order of kn. Compared to the QR algorithm, this is a much more efficient way of extracting a few eigenvalues of A provided: (i) k can be kept small with respect to n; (ii) it is cheap to perform the matrix vector multiplication. The precise definition of the Arnoldi method will be given in Section 1.3, and the rest of the thesis will focus on developing techniques to further improve the Arnoldi method.

We shall mention that when (1.2) is discretized by a finite element method, the algebraic eigenvalue problem takes the form

$$Kx = \lambda Mx, \tag{1.4}$$

where M is likely to be sparse but not necessarily diagonal. This variation is often referred to as a *generalized eigenvalue problem*. In structural mechanics, K is called a *stiffness* matrix, and M a *mass* matrix. Generalized eigenvalue problems also arise from the following type of ordinary differential equation:

$$M\frac{d^2u}{dt^2} + Ku = f.$$

For small size K and M, the QZ algorithm [49] is usually the method of choice for computing the full spectrum of the matrix pencil (K, M). In the large-scale setting, one often transforms (1.4) into a standard eigenvalue problem (1.1) before applying the Arnoldi method. The computation of generalized eigenvalues and eigenvectors will be addressed in Chapters 4 and 6.

1.2 The Power Method

The power method described in Figure 1.1 forms the basis of many numerical procedures for eigenvalue computation. The scaling in Step 1.4 of the algorithm is used

Power Method

Input: matrix A, a starting vector v.
Output: an eigenvalue λ and the corresponding eigenvector x

1. for $j = 1, 2, \ldots$ **until** convergence
  1.1. $w \leftarrow Av$;
  1.2. $\lambda = \frac{v^H w}{v^H v}$;
  1.3. $\beta \leftarrow$ max element of w;
  1.4. $v \leftarrow w/\beta$;
2. end;
3. $x \leftarrow v$;

Figure 1.1 The power iteration.

to avoid an eventual overflow or underflow. Notice that this scaling is homogeneous. That is, if w is multiplied by a scalar α, the power method yields the same v. The superscript "H" that appears in Step 1.2 denotes the complex conjugate transpose.

We assume for ease of presentation that A has n distinct eigenvalues and they can be ordered such that

$$|\lambda_1| < |\lambda_2| < \cdots < |\lambda_n|.$$

With this assumption, we can expand the starting vector v in the eigenvector basis $\{x_j\}_{j=1}^n$,

$$v = \sum_{j=1}^{n} \gamma_j x_j.$$

If $\gamma_n \neq 0$, the power method produces a sequence of vectors

$$v^{(k)} = \tau_k A^k v = \tau_k \gamma_n \lambda_n^k \Big[x_n + \sum_{j=1}^{n-1} \Big(\frac{\gamma_j}{\gamma_n}\Big) \Big(\frac{\lambda_j}{\lambda_n}\Big)^k x_j \Big],$$

where τ_k is an accumulated scaling factor. With a proper scaling strategy, $v^{(k)}$ converges to x_n at the rate of λ_{n-1}/λ_n. A few interesting observations of the power method are:

- Step 1.2 in Figure 1.1 can be viewed as the action of projecting A into a 1-dimensional subspace spanned by v. The power method repeatedly improves this subspace by applying a polynomial in A to v. This polynomial has the form $p(\lambda) = \gamma\lambda^m$. Applying $p(A)$ to v "filters" out the unwanted eigen-components from v.

- If we replace A with $(A - \mu I)^{-1}$, the shifted and inverted power method (sometimes referred to as the *inverse iteration*) converges to an eigenvector associated with the eigenvalue nearest to μ.

- If we allow μ to change from step to step, a rapid convergence may occur. In particular, if we let μ be the Rayleigh Quotient $v^H Av/(v^H v)$, the iteration converges quadratically in general, and cubically if A is Hermitian. This modification of the power method is often referred to as the *Rayleigh Quotient Iteration* (RQI.) Although it is not obvious within the present context, RQI is actually closely related to applying Newton's method to solve $F(x) = Ax - \lambda(x)x = 0$, where $\lambda(x) = x^H Ax/x^H x$.

- If the starting vector v does not contain a component in the direction of x_n and $\gamma_{n-1} \neq 0$, then the power iteration will converge to x_{n-1} at the rate of $\lambda_{n-2}/\lambda_{n-1}$. This observation leads to a simple mechanism for computing more than one eigenpair of A. After obtaining x_n, one may apply the power method to the *deflated* matrix $(I - x_n x_n^H)A(I - x_n x_n^H)$ to expose the next eigenpair.

These observations lead to some general principles which we will follow to develop acceleration schemes for the Arnoldi iteration. In particular, the idea of polynomial filtering is utilized in Chapter 2. Bringing an inverse iteration-like convergence to the Arnoldi iteration forms the main motivation for the method we will introduce in Chapter 3. The technique of "shift and invert" is the main subject of Chapter 4, and acceleration strategies based on Newton correction are examined in Chapter 6.

1.3 The Arnoldi Method

The main drawback of the power method is that it computes only one eigenvalue and eigenvector at a time. One can easily generalize the power method to a subspace iteration in which A is repeatedly applied to a basis of a k-dimensional subspace [57]. However, it has been observed that this method often converges slowly, especially when the gap between λ_k and λ_{k+1} is small.

Better approximations to the eigenvalues and eigenvectors of A can be drawn from a *Krylov subspace*

$$\mathcal{K}(A, v_0; k) = \{v_0, Av_0, ..., A^{k-1}v_0\}.$$

The construction of an orthonormal basis of $\mathcal{K}(A, v_0; k)$ yields the *Arnoldi method* which is often characterized by the matrix equation

$$AV_k = V_k H_k + f_k e_k^T, \quad V_k^H V_k = I_k, \quad V_k^T f_k = 0. \tag{1.5}$$

The columns of $V_k \in \mathbb{C}^{n\times k}$ and $H_k \in \mathbb{C}^{k\times k}$ are generated by the Gram-Schmidt procedure outlined in Figure 1.2. The matrix H_k is a representation of the projection

Arnoldi Iteration

Input: $(A,\ v_0)$
Output: $(V_k,\ H_k,\ f_k)$ such that $AV_k = V_kH_k + f_ke_k^T$, $V_k^TV_k = I_k$, and $V_k^Tf_k = 0$.

1. $v_1 \leftarrow v_1/\|v_1\|$;
2. $w \leftarrow Av$; $\alpha_1 = v_1^Hw$;
3. $H_1 = (\alpha_1)$; $V_1 = (v_1)$; $f_1 \leftarrow w - v_1\alpha_1$;
4. **for** $j = 1, 2, 3, ..., k-1$
  4.1. $\beta_j = \|f\|$; $v_{j+1} \leftarrow f/\beta_j$
  4.2. $V_{j+1} = (V_j,\ v_{j+1})$; $H_j \leftarrow \begin{pmatrix} H_j \\ \beta_j e_j^T \end{pmatrix}$;
  4.3. $z \leftarrow Av_{j+1}$;
  4.4. $h \leftarrow V_j^Hz$; $H_{j+1} = (H_j,\ h)$;
  4.5. $f_j \leftarrow z - V_{j+1}h$;
5. **end**;

Figure 1.2 The Arnoldi iteration.

of A into $\mathcal{K}(A, v_0; k)$. Notice that the procedure described in Figure 1.2 does not transform A in any way, therefore it does not require A to be explicitly stored. One only needs to provide the operation of a matrix-vector multiplication to carry out the Arnoldi process.

Approximate eigenpairs (θ_j, z_j), where $z_j = V_jy_j$, are obtained from the Galerkin condition:

$$V_k^H(AV_ky_j - \theta_jV_ky_j) = H_ky_j - \theta_jy_j = 0.$$

The eigenvalues of H_k are referred to as *Ritz values*, and the eigenvector approximations, $z_j = V_jy_j$ $(j = 1, 2, ..., k)$, are referred to as *Ritz vectors*. We call equation

(1.5) an *Arnoldi factorization* and the columns of V_k *Arnoldi vectors*. When A is Hermitian, the matrix H_k becomes tridiagonal, and the Arnoldi process reduces to the *Lanczos method* [37]. In this thesis, we will use the term Arnoldi and Lanczos interchangeably when symmetric eigenvalue problems are considered.

The following identity

$$\begin{aligned} \frac{\|Az - \theta z\|}{\|A\|} &= \frac{\|AV_k y - \theta V_k y\|}{\|A\|} \\ &= \frac{\|(V_k H_k y + f_k e_k^T)y - \theta V_k y\|}{\|A\|} \\ &= \frac{\|f_k\||e_k^T y|}{\|A\|} \end{aligned}$$

indicates that one can check the accuracy of the Ritz approximation without explicitly calculating the residual $r = Az - \theta z$. We will refer to the quantity $\|f_k\||e_k^T y|$ as the *Ritz error estimate*.

Let us denote the spectrum of A by $\sigma(A)$. It is well known that Ritz values lie in the convex hull of $\sigma(A)$, and the well separated extremal eigenvalues of A emerge rapidly in the Arnoldi process [33, 54, 70, 71]. Here, extremal eigenvalues refer to eigenvalues nearest the vertices of the convex hull of $\sigma(A)$. When the desired eigenvalues do not coincide with these extremal eigenvalues, a Krylov subspace of large dimension may be required to provide satisfactory approximations. This often results in a significant storage for V_k and enormous effort to maintain $V_k^H V_k = I_k$ numerically. To avoid this overhead, additional strategies for accelerating the Arnoldi method within a subspace of limited size are required.

1.4 Organization of the Thesis

This thesis is aimed at providing theoretical foundations and practical guidelines for accelerating the Arnoldi iteration. Several types of acceleration schemes shall be

investigated. The first scheme is based on the idea of modifying the starting vector v_0 and repeating the k-step Arnoldi process until V_k eventually converges to a proper subspace. The starting vector is usually modified by applying a function of A, $\psi(A)$, to v_0 so that the contribution of the unwanted eigencomponents of A to $\mathcal{K}(v_0, A; k)$ is attenuated. Common choices for $\psi(\lambda)$ are polynomials or rational functions. This approach is known as the method of *restarting* an Arnoldi iteration. When $\psi(\lambda)$ is a polynomial, we call the restarting strategy a *polynomial restart*. Likewise, if $\psi(\lambda)$ is a rational function, we used the term *rational restart*. The second acceleration scheme transforms the original eigenvalue problem into one whose extreme eigenvalues can be easily identified and mapped back to the desired eigenvalues of the original problem. It is often referred to as the method of *spectral transformation*. One can also treat the eigenvalue problem as solving a nonlinear equation (in x) upon which Newton type methods may be applied. The recent successes in this area are highlighted by the Jacobi-Davidson Method [79].

We will focus on polynomial restarting in Chapter 2, and review the Implicitly Restarted Arnoldi (IRA) algorithm [81]. Chapter 3 is devoted to the discussion of a rational restarting strategy which we call the *Truncated RQ* (TRQ) iteration [84]. The technique of spectral transformation through LU factorization is examined in Chapter 4, and efficient polynomial spectral transformations are proposed in Chapter 5. Finally, Newton based acceleration schemes are discussed in Chapter 6. Numerical examples are provided throughout the thesis to demonstrate the effectiveness of various acceleration strategies.

1.5 Notation

Throughout this thesis, capital and lower case Latin letters usually denote matrices and vectors respectively, while lower case Greek letters denote scalars. The j-th

canonical basis vector is denoted by e_j. The Euclidean norm is used exclusively and is denoted by $\|\cdot\|$. The transpose of a matrix A is denoted by A^T and conjugate transpose by A^H. Upper Hessenberg matrices will appear frequently and are usually denoted by the letter H. Elements from the upper triangle of such a matrix will be denoted by γ_{ij} and the j-th sub-diagonal element will be denoted by $\beta_j = \gamma_{j+1,j}$. The conjugate of a complex number α is denoted by $\bar{\alpha}$.

Chapter 2

Polynomial Restarting and IRA

In Section 1.3, we pointed out only one aspect of the convergence behavior of the Arnoldi method, that is, the extremal eigenvalues tend to appear much earlier in the Arnoldi process than the others. The following lemma illustrates that rapid convergence occurs if we choose the starting vector v_0 of the Arnoldi process carefully.

Lemma 2.1 If $v_0 \in \text{span}\{W_k\}$, where $S_k \in \mathbb{C}^{n\times k}$ and $AW_k = W_k G_k$ for some $G_k \in \mathbb{C}^{k\times k}$. then

$$\mathcal{K}(A, v_0; k) \subset \text{span}\{W_k\},$$

and an Arnoldi process with v_0 as the starting vector terminates in k or fewer steps.

Proof Since $v_0 \in \text{span}\{W_k\}$, $v_0 = W_k t$ for some $t \in \mathbb{C}^{k\times 1}$. It follows that

$$A^j v_0 = A^j W_k t = W_k G_k^j t = W_k g \in \text{span}\{W_k\},$$

where $g = G_k^j t$. Thus, $\mathcal{K}(A, v_0; k) \subset \text{span}\{W_k\}$.

If the Arnoldi process with the starting vector v_0 does not terminate in k steps, then there exist V_k, H_k and f_k such that

$$AV_k = V_k H_k + f_k e_k^T, \quad V_k^H V_k = I_k, \quad V_k^H f_k = 0,$$

and the Hessenberg matrix H_k is unreduced. However, since

$$\text{span}\{V_k\} = \mathcal{K}(A, v_0; k) \subset \text{span}\{W_k\},$$

it follows that $AV_k \in \text{span}\{W_k\}$. Therefore f_k must lie in $\text{span}\{W_k\}$ as well. Since $V_k^H V_k = I_k$ and $V_k^H f_k = 0$. There are $k+1$ basis vectors for a k-dimensional subspace, a contradiction. Therefore the Arnoldi factorization must terminate in k or fewer steps. □

If the starting vector v_0 belongs to a subspace spanned by k desired eigenvectors, then at least in exact arithmetic the Arnoldi factorization terminates in k steps giving

$$AV_k = V_k H_k.$$

The eigenvalues of H_k are exactly the eigenvalues of A and the Ritz vectors are the eigenvectors of A. However, in practice one rarely finds such a good v_0 before the eigenvalue problem is solved. Consequently, one may have to generate a large Krylov subspace to obtain satisfactory approximations to the desired eigenvalues and eigenvectors. It is important to keep the Arnoldi vectors mutually orthogonal. Loss of orthogonality amongst these vectors may result in serious contamination of the computed eigenvalues. Unfortunately, maintaining orthogonality takes enormous amount of storage and computational effort when the Krylov subspace generated is large.

To reduce the cost and thus improve the speed of convergence, one shall avoid building a large Krylov subspace. The alternative is to initially carry out only k steps of an Arnoldi iteration, where k is a moderate value. If the k-dimensional Krylov subspace fails to provide accurate approximations to the desired eigenpairs, we recompute a k-step Arnoldi factorization using a modified starting vector. The modification of v_0 typically takes advantage of the eigenvector approximations obtained in the previous Arnoldi run and tries to remove the unwanted eigencomponents from v_0. This modification and refactorization procedure forms the basic pattern of a *restarted* Arnoldi

iteration. It is repeated until the unwanted the eigencomponents in v_0 become insignificant.

There are a number of ways to implement a restart. A simple restarting strategy is to take v_0 as a linear combination of Ritz vectors associated with the desired Ritz values [50]. A more sophisticated approach is to replace v_0 with $\psi(A)v_0$, where $\psi(\lambda)$ is a function constructed to filter out the unwanted eigencomponents from v_0. In this chapter, we focus on the case in which $\psi(\lambda)$ is a polynomial.

Saad [73] proposed restarting the Arnoldi process by explicitly computing

$$v_0 \leftarrow \psi(A)v_0,$$

where $\psi(\lambda)$ is a desired polynomial. If the degree of the polynomial is p, then p matrix-vector multiplications are required to restart the Arnoldi process in addition to the k matrix-vector multiplications used in the Arnoldi iteration.

The Implicitly Restarted Arnoldi (IRA) algorithm proposed by Sorensen [81] avoids computing $\psi(A)v_0$ directly. The new starting vector emerges as a by-product of a sequence of implicit QR-updates. Besides producing a new starting vector, these QR updates also construct, implicitly, a new Arnoldi factorization of length $k-p$. Only p ($p < k$) matrix-vector multiplications are needed thereafter to complete a new Arnoldi sweep.

The theory on the implicitly restarted Arnoldi method is fully developed in [81]. Its relationship with a truncated QR iteration is also exploited in [38]. The best way to explain IRA is probably to first introduce the implicitly shifted QR iteration, a "dense" algorithm that requires a full Hessenberg reduction and a sequence of unitary similarity transformations, then show that IRA is just a clever way of truncating the QR iteration. The advantage of this approach is that the convergence of IRA can be investigated in the same way the QR algorithm is analyzed.

The purpose of restarting is to filter out the unwanted eigencomponents from v_0. The quality of the filter depends on the shifts chosen during the QR updates. We examine three different shifting strategies in Section 2.4.1. Numerical examples are presented in Section 2.4 to demonstrate the effectiveness of the polynomial restart.

2.1 The Implicitly Shifted QR Algorithm

The QR algorithm for the dense matrix eigenvalue calculation begins with a full Hessenberg reduction

$$AV = VH, \tag{2.1}$$

where $V^H V = I$ and $H \in \mathbb{C}^{n\times n}$ is upper Hessenberg. The reduction is followed by the application of a sequence of unitary similarity transformations to H. That is, we set

$$H_0 = H, \quad H_{j+1} = Q_j^H H_j Q_j.$$

This process eventually drives H_j into an upper triangular form. Each Q_j comes from the QR decomposition of $H_j - \mu_j I$ for some shift μ_j. That is, Q_j satisfies

$$H_j - \mu_j I = Q_j R_j.$$

To simplify our discussion, let's first examine what happens during one step of the QR iteration, and drop the subscripts of μ, Q, R and H for clarity. After rewriting (2.1) as

$$(A - \mu I)V = V(H - \mu I),$$

we can utilize the QR decomposition of $H - \mu I$ to obtain

$$(A - \mu I)V = VQR, \tag{2.2}$$

Postmultiplying (2.2) by Q yields

$$(A - \mu I)(VQ) = (VQ)(RQ). \tag{2.3}$$

We can deduce, after adding the shift back to (2.3) and using the fact

$$RQ + \mu I = Q^H(H - \mu I)Q + \mu I = Q^H HQ,$$

that

$$A(VQ) = (VQ)(Q^H HQ). \tag{2.4}$$

Due to the Hessenberg structure of H, $Q^H HQ$ remains upper Hessenberg. If we let $V_+ \equiv VQ$ and $H_+ \equiv Q^H HQ$, then a new Hessenberg reduction

$$AV_+ = V_+ H_+,$$

follows. It is easy to see from (2.2) that the first column of the updated basis V_+ is related to that of V through the following equation

$$(A - \mu I)v_1 = v_1^+ \rho_{1,1},$$

where $v_1 = Ve_1$, $v_1^+ = V_+ e_1$ and $\rho_{1,1} = e_1^T R e_1$. That is, the vector v_1^+ is what we would have obtained by applying one step of the shifted power iteration to v_1.

With a carefully selected set of shifts $\{\mu_j\}$, we repeat the procedure described by equations (2.2)–(2.4) until H becomes upper triangular. It follows from the above discussion that the QR iteration can be viewed as a sequence of implicitly restarted Hessenberg reductions. At the end of the p-th QR-iteration, the new starting vector v_1^+ is related to the original starting vector v_1 via

$$v_1^+ = \psi(A)v_1, \tag{2.5}$$

where

$$\psi(\lambda) = (\lambda - \mu_1)(\lambda - \mu_2)\cdots(\lambda - \mu_p).$$

We will refer to $\psi(\lambda)$ as a *cumulative polynomial*. In practice, the initial Hessenberg reduction is accomplished by applying a sequence of Householder transformations to A. The QR updates (2.2)–(2.4) are carried out using a "bulge chase" mechanism [22, 23].

2.2 Implicitly Restarted Arnoldi

In the large-scale setting, a full reduction to a Hessenberg form via the Householder transformation is usually intractable. A partial reduction can be constructed by the Arnoldi factorization

$$AV_m = V_m H_m + f_m e_m^T, \tag{2.6}$$

where $m = k + p$. Since a QR update starts from the upper left corner of H, one can mimic, within the truncated Hessenberg reduction, the same updating process that takes place in a full QR iteration.

Given a shift μ, one factors $H_m - \mu I$ into a product

$$H_m - \mu I = Q_m R_m,$$

where Q_m is unitary and R_m upper triangular. The same techniques described in (2.2)–(2.4) can be used to obtain

$$A(V_m Q_m) = (V_m Q_m)(Q_m^H H_m Q_m) + f_m e_m^T Q_m. \tag{2.7}$$

Note that due to the Hessenberg structure of Q_m, only the last two columns of the matrix $f_m e_m^T Q_m$ are nonzero. This implies that the first $m-1$ columns of (2.7) satisfy a new Arnoldi factorization implicitly constructed by the previous QR update. After repeating this process for the next $p-1$ shifts, we obtain

$$A(V_m Q) = (V_m Q)(Q^H H_m Q) + f_m e_m^T Q, \tag{2.8}$$

where Q is the accumulation of p unitary factors generated by the previous QR updates. Again, it follows from the Hessenberg structure of each unitary factor that the last row of Q has only $p+1$ nonzero. Consequently, the first $k(=m-p)$ columns of $f_m e_m^T Q$ remain zero columns, indicating that the first k columns of (2.8) form a new Arnoldi factorization. One can extend this factorization to length $k+p$ by performing

p standard Arnoldi steps. Since $m = k + p$ is kept small, one can afford to use the Daniel, Gragg, Kaufman and Stewart (DGKS) algorithm [14], which performs the correction (2.9) if necessary,

$$c = V_{j+1}^H f_{j+1}; \quad f_{j+1} \leftarrow f_{j+1} - V_{j+1} c; h \leftarrow h + c; \tag{2.9}$$

to maintain full orthogonality among all Arnoldi vectors. Just like the full QR iteration, the first column of the new Arnoldi basis is related to that of the original one via

$$v_1^+ = \psi(A) v_1,$$

where $\psi(\lambda) = (\lambda - \mu_1)(\lambda - \mu_2) \cdots (\lambda - \mu_p)$. Figure 2.1 shows the four stages of an IRA iteration. We will use the notation `IRA(k,p)` later to denote an IRA iteration that computes k eigenvalues and applies p shifts during each sweep.

2.3 Choice of shifts

In view of (2.5), the choice of shifts $\{\mu_j\}$ in IRA plays an important role in the construction of a polynomial that "filters out" unwanted components from the starting vector. A number of options are available. One may compute eigenvalues of the current H_m, and sort them into two disjoint sets Ω_w and Ω_u. The set Ω_w includes k "wanted" Ritz values, whereas all Ritz values in Ω_u are regarded as "unwanted". The unwanted Ritz values are used as shifts. They are often referred to as the *exact shifts*. By placing the zeros of the polynomial at these shifts, one hopes to reduce the contribution of unwanted eigencomponents to the starting vector. This shifting scheme is proposed in [81] and implemented in `ARPACK` [40]. It is optimal in the sense that it makes the best use of the spectral information obtained from the current Arnoldi run.

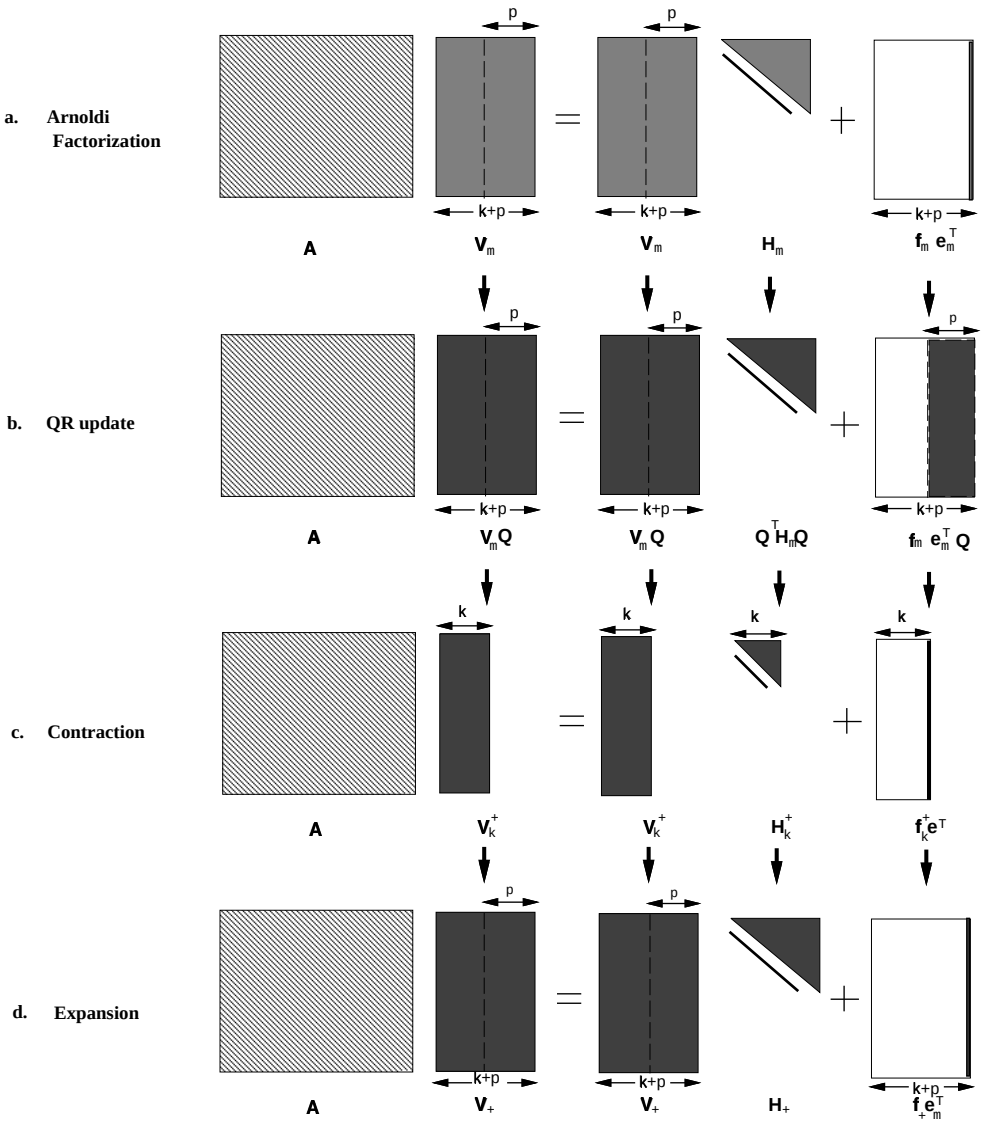

Figure 2.1 One cycle of the Implicitly Restarted Arnoldi Iteration. The shaded area contains nonzero entries. The Arnoldi factorization (a) is modified by a QR update (b) which produces a new factorization of length k (c). The new factorization is extended to the last column of V_m and H_m by running p additional Arnoldi iterations (d).

Other shifting strategies are possible. For example, if all unwanted eigenvalues of A are contained in an interval $[\alpha, \beta]$, it is natural to consider a filtering polynomial that is damped uniformly within $[\alpha, \beta]$. A shifted and scaled Chebyshev polynomial

$$C_p(\lambda; \alpha, \beta) = T_p\Big(\frac{\alpha + \beta - 2\lambda}{\alpha - \beta}\Big),$$

where

$$T_p(\lambda) = \begin{cases} \cos(p \cos^{-1}(\lambda)), & |\lambda| \leq 1 \\ \cosh(p \cosh^{-1}(\lambda)), & |\lambda| > 1 \end{cases} \tag{2.10}$$

satisfies this requirement. The roots of the Chebyshev polynomial are explicitly known:

$$\lambda_j = \frac{\alpha + \beta - 2\tilde{\lambda}_j}{\alpha - \beta}, \quad \text{where} \quad \tilde{\lambda}_j = \cos\Big[\frac{(2j-1)\pi}{2p}\Big], \quad j = 1, ..., p.$$

We will refer to shifts chosen in this fashion as *Chebyshev shifts*. In practice, the values of α and β are often estimated from the Ritz values. For example, if the k smallest eigenvalues of a hermitian matrix A are of interest, one can choose $\alpha = \theta_{k+1}$ and $\beta = \theta_{k+p}$, where

$$\theta_1 < \theta_2 \cdots < \theta_{k+1} < \cdots < \theta_{k+p}$$

are Ritz values computed in the current Arnoldi(Lanczos) run.

Another interesting set of shifts are the *Leja* points. The Leja points $\{z_j\}$ for a given set $\mathbf{K}$ are defined as follows [9]:

Definition 2.1 Let $w(z)$ be a weight function defined on $\mathbf{K}$. Choose $z_0 \in \mathbf{K}$ such that

$$w(z_0)|z_0| = \max_{z \in \mathbf{K}} w(z)|z|. \tag{2.11}$$

Choose $z_j \in \mathbf{K}$ such that

$$w(z_j) \prod_{\ell=0}^{j-1} |z_j - z_\ell| = \max_{z \in \mathbf{K}} w(z) \prod_{\ell=0}^{j-1} |z - z_\ell|. \tag{2.12}$$

A sequence of points $\{z_j\}$ satisfying (2.11) and (2.12) are called *Leja* points for the set $\mathbf{K}$.

In the context of IRA, one can use the procedure described above to generate Leja shifts for a set $\mathbf{K}$ that contains most of the unwanted eigenvalues. The bounds for set $\mathbf{K}$ are often estimated from the Ritz values. Moreover, one can easily modify the procedure given above to take into account the Leja points generated during the previous IRA sweep. It is reported in [9] that this shifting strategy is extremely helpful when the number of shifts one can apply is limited and the range of the unwanted eigenvalues is large.

2.4 Numerical Examples

Two numerical examples are presented in this section to demonstrate the effectiveness of polynomial restarts. The first example involves a rather simple symmetric matrix. In this case, IRA reduces to the implicitly restarted Lanczos (IRL) algorithm. A few of the lowest eigenvalues are of interest. We wish to use this example to illustrate the advantage of IRL over the standard Lanczos run and the effects of different shifting strategies on the convergence of IRL. Our second example is extracted from the earlier work [90]. The non-symmetric matrix has both real and complex eigenvalues. The rightmost eigenvalues are computed. We wish to use this example to show that the polynomial filtering in IRA can be as effective on the complex plane as on the real line. All experiments were performed on a SUN-Ultra2 in double precision. The FORTRAN package `ARPACK` was called to carry out the IRA iteration.

2.4.1 One-dimensional Laplacian

Comparison of IRL with Standard Lanczos

The matrix $A \in \mathbb{R}^{100\times 100}$ used in this example has the form

$$A = \begin{bmatrix} 2 & -1 & & -1 \\ -1 & 2 & \ddots & \\ & \ddots & \ddots & -1 \\ -1 & & -1 & 2 \end{bmatrix}.$$

It is obtained from the standard central difference discretization of the one-dimensional Laplacian defined on $[0, 1]$ with a periodic boundary condition.

Computing the smallest eigenvalues of A is difficult with the standard Lanczos iteration mainly because the eigenvalues of interest are tightly clustered at the low end of the spectrum. In the meantime, there are many large eigenvalues at the other end of the spectrum.

To see this phenomenon, we first run 90 steps of standard Lanczos. We enforce the orthogonality between all Lanczos vectors by explicitly reorthgonalizing the new Lanczos vector, at each step, against all previously generated Lanczos vectors. The 5 smallest Ritz values are listed in Table 2.1 along with the exact eigenvalues of A. One can easily see that these Ritz values do not completely agree with the exact eigenvalues. In particular, the third and the fifth eigenvalues are only accurate up to the second digit.

To obtain a better approximation, we further increase the number of Lanczos steps. Unlike the previous run in which we saved all Lanczos vectors and kept them mutually orthogonal, we now generate these vectors using a three term recurrence. This approach reduces both storage and the amount of computation needed to complete the Lanczos run. However, there is no guarantee that the Lanczos vector generated at

eigenvalue	Ritz value
0.0	4.2×10^{-16}
3.94654×10^{-3}	3.94654×10^{-3}
3.94654×10^{-3}	3.95503×10^{-3}
1.57705×10^{-2}	1.57705×10^{-2}
1.57705×10^{-2}	1.59375×10^{-2}

Table 2.1 Ritz values computed from a 90-step standard Lanczos run with reorthogonalization.

each step will be orthogonal to the previously generated Lanczos vectors. Moreover, if eigenvectors are needed, one must run the same Lanczos process again to regenerate all the Lanczos vectors and to accumulate them to form the Ritz vectors. The Ritz values computed after 120 steps of this Lanczos run are listed in Table 2.2. Notice that the second and third eigenvalue approximations are spurious copies of the first eigenvalue. These spurious copies are caused by the loss of orthogonality between the Lanczos vectors. Without knowing the multiplicity of each eigenvalue in advance, it is rather difficult to tell whether a computed eigenvalue is a spurious copy. (Cullumn and Wilboughby proposed some mechanisms ghost to detect and eliminate the spurious eigenvalues [13]. One may also use a selective reorthogonalization scheme [58] to avoid introducing the spurious eigenvalues.)

Note that we purposely set the number of Lanczos steps larger than the dimension of the matrix so that the number of matrix-vector multiplications ($120 \times 2 = 240$) performed here (to acquire both Ritz values and Ritz vectors) is comparable to what we will use in an IRL run.

Finally, we apply `IRL(5,20)` to A. (Recall `IRL(5,20)` stands for an IRL run with $k = 5$, $p = 20$. The dimension of the Krylov subspace constructed is $k + p = 25$.) The convergence threshold is set at 10^{-8}. The exact shifting strategy is used. All of

eigenvalue	Ritz value
0.0	1.3×10^{-16}
3.94654×10^{-3}	6.8×10^{-15}
3.94654×10^{-3}	1.7×10^{-9}
1.57705×10^{-2}	3.94654×10^{-3}
1.57705×10^{-2}	3.94654×10^{-3}

Table 2.2 Ritz values computed from a 120-step standard Lanczos run with no reorthogonalization.

the five smallest eigenvalues are captured in 17 restarts. The entire computation uses 235 matrix-vector multiplications (MATVECs). The reliability and efficiency of IRL are evident in this experiment.

The Effect of Shifting Strategies

In the next experiment, we show that using a different shifting strategy can affect the convergence of IRL. The exact, Chebyshev and Leja shifting strategies are considered and compared. To generate the Chebyshev or Leja shifts, one must provide the

Shifts	MATVECs
exact	235
Chebyshev	380
Leja	285

Table 2.3 Comparison of three shifting strategies

upper (β) and lower (α) bounds of an interval that contains most of the unwanted eigenvalues of A. In this experiment, we set $\beta = 4.1$, an upper bound of the spectrum, and set $\alpha = \theta_{k+1}$, the $(k+1)$st Ritz value obtained in each IRL sweep. We observe from Table 2.3 that the exact shifting yields the best performance among the three. The filter polynomial associated with the exact shifts is plotted in Figure 2.2. One

clearly sees that the polynomial maps the large eigenvalues to zero, removing the contribution of the corresponding eigenvector from the starting vector of a Lanczos iteration. We shall point out that when Chebyshev or Leja points are used, the

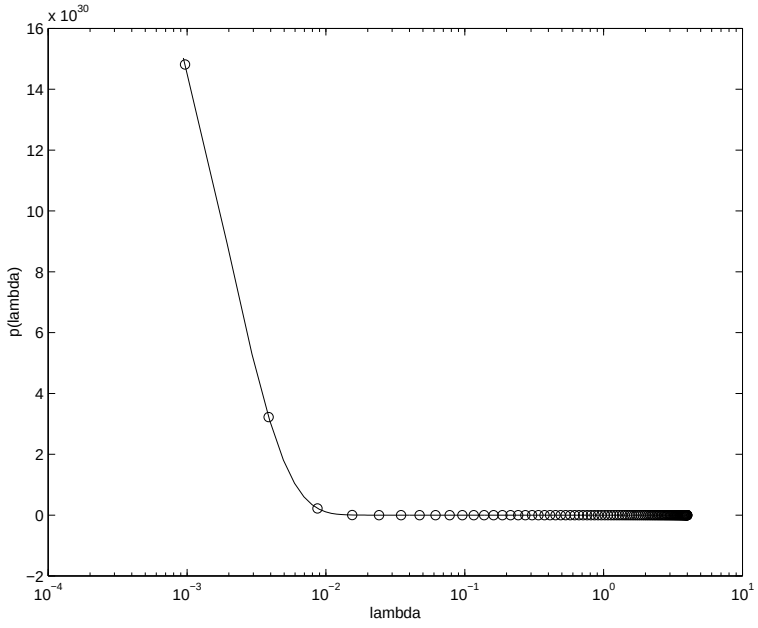

Figure 2.2 The filtering polynomial produced by IRL with exact shifts. The circles denote $p(\lambda_j)$, where λ_j $(j = 2, 2, ..., 100)$ are eigenvalues of A.

performance of the computation is sensitive to the value of α. One may improve the performance of an IRL run by making α slightly larger than θ_{k+1}. Table 2.4 illustrates this observation by listing the performance of Chebyshev shifts generated with different α values. These α's are set to be the $(k+j)$-th $(j = 1, 2, ...)$ Ritz values obtained in each IRL sweep. We observe that for this particular problem, $\alpha = \theta_{k+3}$ provides the best performance.

The performance of the shifting strategy is also affected by the number of shifts (p) allowed in IRL iteration. We show in the following experiment that when p is small, Leja shifts can be very effective. Table 2.5 shows the number of MATVECs

α	MATVECS
θ_{k+1}	380
θ_{k+2}	350
θ_{k+3}	275
θ_{k+4}	290
θ_{k+5}	320

Table 2.4 Performance of Chebyshev shifts with different α values.

used in IRL runs associated with all three shifting strategies and different values of p. Apparently, when $p = 5$, the Leja shifts perform much better than either the exact

p	exact	Chebyshev	Leja
5	409	465	280
6	366	497	293
7	337	390	278
8	312	413	285
9	301	410	311
10	284	385	265

Table 2.5 Comparison of matrix-vector multiplications for three shifting strategies with small p values.

or the Chebyshev shifts. There is a heuristic explanation for this phenomenon. Since a small value of p results in a low degree filtering polynomial, only a few unwanted eigencomponents can be removed in each IRL sweep. Consequently, Ritz values do not vary much from one IRL sweep to another. This causes the roots of the cumulative filtering polynomial to cluster around a few locations within $[\alpha, \beta]$. This phenomenon does not occur with Leja shifts because the selection of new Leja points takes into account the previously generated shifts. The roots of cumulative polynomials are more uniformly distributed within $[\alpha, \beta]$, and thus provide more effective damping. Figure

2.3 demonstrates the difference between these two shifting strategies by plotting the roots of the cumulative polynomials at each IRL sweep.

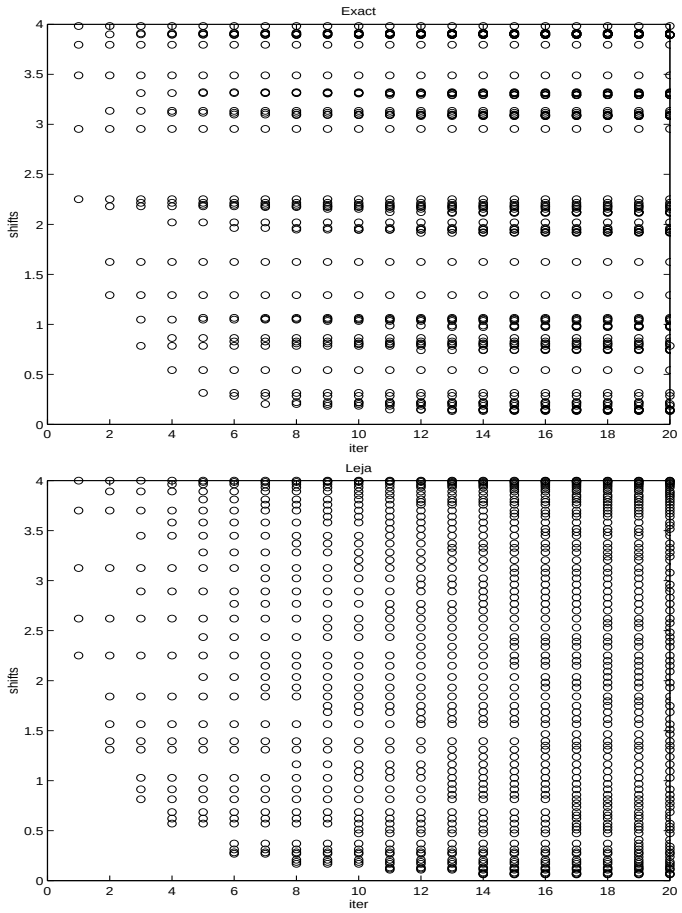

Figure 2.3 Distribution of the roots of the cumulative filtering polynomials at each IRL sweep. The top plot was generated from the exact shifts applied during the IRL iteration. The bottom plot corresponds to IRL with Leja points.

2.4.2 Crystal Growth

Let us now look at a non-symmetric example that is more interesting and challenging. The problem arises from the linear stability analysis of crystal growth or dendritic

solidification, a rapid and complicated process that occurs when one places some crystal of pure substance into an undercooled bath of its liquid phase. The goal of the analysis is to understand the temperature distribution of the material during the solidification and the time-dependent movement of the interface between its solid and liquid phase [30, 51, 52, 36, 47, 8]

It is well known that the temperature distributions in both the liquid and solid phase, U_l and U_s, satisfy the dual diffusion equations

$$\frac{\partial U_l}{\partial t} = \alpha \nabla^2 U_l, \quad \frac{\partial U_s}{\partial t} = \alpha \nabla^2 U_s,$$

where the constant α is related to thermal diffusivity. At the solid-liquid interface the displacement of the moving front $\vec{r}$ obeys the conservation law:

$$\frac{d\vec{r}}{dt} \cdot \vec{n} = \alpha(\nabla U_s \cdot \vec{n} - \nabla U_l \cdot \vec{n}).$$

Here $\vec{n}$ denotes the unit outward normal to the interface pointing from the solid phase to the liquid phase. The evolution of the solid front also depends on its geometric properties and other initial and boundary conditions.

For a simple model in which the initial interface is of parabolic shape, one can write down an exact steady-state solution after making use of some coordinate transformation and change of variables [90].

The time-dependent behavior of the solidification process can be analyzed by seeking admissible time-dependent perturbations of the form $\hat{U}e^{\sigma t}$, where $\hat{U}$ is time-independent and σ represents the growth rate of the perturbation. Interesting perturbations are those associated with a positive growth rate (indicating that the steady-state solution is unstable.)

A set of new equations for the perturbation can be derived by substituting the perturbed temperature distribution and interface displacement into the governing diffusion equation (or convection diffusion in the new coordinate system) and linearizing

the resulting equations at the steady state. After some additional changes of variables and simplifications, we arrive at an eigenvalue problem

$$a(x,y)\frac{\partial^2 U}{\partial x^2} + b(x,y)\frac{\partial^2 U}{\partial y^2} + c(x,y)\frac{\partial U}{\partial x} + d(x,y)\frac{\partial U}{\partial y} = \lambda U, \tag{2.13}$$

$$e(x,y)\frac{\partial U}{\partial x} + f(x,y)N + g(x,y)\frac{\partial U}{\partial y} = \lambda N, \tag{2.14}$$

where U is represents the perturbation of the temperature distribution, N represents the perturbation of the interface displacement, and the eigenvalue λ is proportional to the growth rate of the perturbation. Because we used some changes of variables to map the original PDE defined on an infinite domain to a finite box, the eigenvalue problem is now defined on the unit square $[0,1]\times[1,2]$. Note that the second equation (2.14) only applies to the temperature distribution on the initial solid-liquid boundary ($y = 1$) on which U and N are related through

$$U = 2pN, \quad \text{where } p \text{ is a dimensionless Peclet number.} \tag{2.15}$$

Other boundary conditions imposed are:

$$\begin{aligned} \text{at } x = 0 \quad &: \quad \frac{\partial U}{\partial x} = 0, \\ \text{at } x = 1 \quad &: \quad \frac{\partial U}{\partial x} = 0, \\ \text{at } y = 2 \quad &: \quad U(x,2) = 0. \end{aligned}$$

The coefficients a, b, c, d, e, f, g in (2.14) are defined as follows:

$$\begin{aligned} a(x,y) &= \frac{(1-x)^4}{\left(\frac{x}{1-x}\right)^2 + \left(\frac{y}{2-y}\right)^2}, \\ b(x,y) &= \frac{(2-y)^4}{4\left(\frac{x}{1-x}\right)^2 + \left(\frac{y}{2-y}\right)^2}, \\ c(x,y) &= -\frac{2(1-x)^3 + 2px(1-x)}{\left(\frac{x}{1-x}\right)^2 + \left(\frac{y}{2-y}\right)^2}, \end{aligned}$$

$$
\begin{aligned}
d(x,y) &= \frac{-\frac{(2-y)^3}{2} + py(2-y)}{\left(\frac{x}{1-x}\right)^2 + \left(\frac{y}{2-y}\right)^2}, \\
e(x,y) &= \frac{p}{1+\left(\frac{x}{1-x}\right)^2}, \\
f(x,y) &= \frac{4p^2+2p}{1+\left(\frac{x}{1-x}\right)^2}, \\
g(x,y) &= \frac{(1-x)^2}{1+\left(\frac{x}{1-x}\right)^2}.
\end{aligned}
$$

Let us choose an appropriate mesh size $\Delta x \times \Delta y$, and define $x_i = i\Delta x$, $y_i = i\Delta y$, $U_{i,j} = U(x_i, y_j)$, and $N_i = N(x_i)$ $(i,j = 1,2,...,n)$. By using the standard second-order finite difference discretization, (to be precise, central difference for all derivatives except $\partial U/\partial y$ in the second equation for which an upwind difference scheme

$$\frac{1}{\Delta y}\Big(-\frac{3}{2}U_{i,0} + 2U_{i,1} - \frac{1}{2}U_{i,2}\Big)$$

is used) we obtain an algebraic eigenvalue problem $Az = \lambda z$, where the matrix A is very sparse. Its structure is shown in Figure 2.4. The eigenvector z consists of the (scaled) temperature distribution on the grid points (x_i, y_j). That is,

$$
z = \begin{pmatrix} \hat{U}_2 \\ \hat{U}_3 \\ \vdots \\ \hat{U}_n \\ \hat{U}_1/(2p) \end{pmatrix} \quad \text{where} \quad \hat{U}_j = \begin{pmatrix} U_{1,j} \\ U_{2,j} \\ \vdots \\ U_{n-1,j} \\ U_{n,j} \end{pmatrix}.
$$

Notice the last component of z also represents the interface perturbation N which is related to the temperature by (2.15).

Since we are interested in perturbations that increase exponentially, eigenvalues whose real parts are positive are of interest. However, these eigenvalues are neither

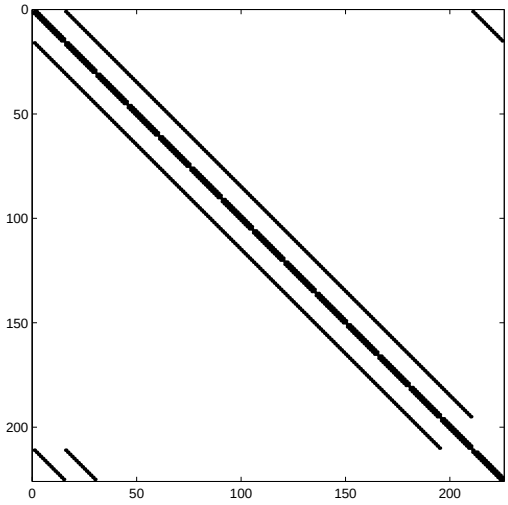

Figure 2.4 The sparsity pattern of A.

well separated from the rest of the spectrum nor dominant as shown in Figure 2.5. The negative eigenvalues at the left end of the spectrum tend to emerge rapidly in a standard Arnoldi iteration. They prevent the desired eigenvalues from converging. The major task of implicit restart is to filter out from the starting vector the eigenvector components associated with these eigenvalues. We set $k = 20$, $p = 30$, and use `ARPACK` to find the 20 eigenvalues with the largest real parts. Exact shifts are used. That is, we computed 50 Ritz values before each restart, kept 20 Ritz values that have the largest real parts, and used the rest as the zeros of a filter polynomial. Since $p = 30$, a polynomial of degree 30 is applied to the starting vector each time a restart is issued. Plotted in Figure 2.6 is the polynomial constructed during the last restart. The polynomial is only shown in the interesting region $[0, 3.5] \times [-2, 2]$ on the complex plain. The exact eigenvalues are marked with pluses, and the computed with circles. The same polynomial is shown in Figure 2.7 on a different region $[-0.2, 1.2] \times [-0.5, 0.5]$. Unfortunately, we are not able to show the entire polynomial

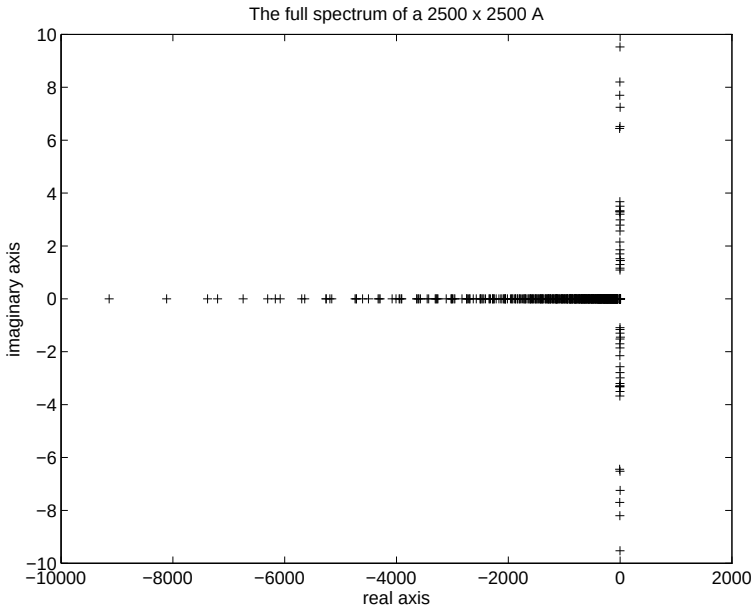

Figure 2.5 The spectrum of A.

on the convex hull of the spectrum since it requires generating too many grid points. We observe from Figure 2.6 that `ARPACK` captured all the desired eigenvalues. This particular IRA run costs 4047 matrix-vector multiplications and 107 CPU seconds. As the dimension of the matrix becomes larger, acceleration using restarting will become more difficult. Techniques introduced in the subsequent chapters may be used to improved the convergence rate of the Arnoldi iteration. We refer readers interested in the computed eigenvalues and eigenvectors to the report [90].

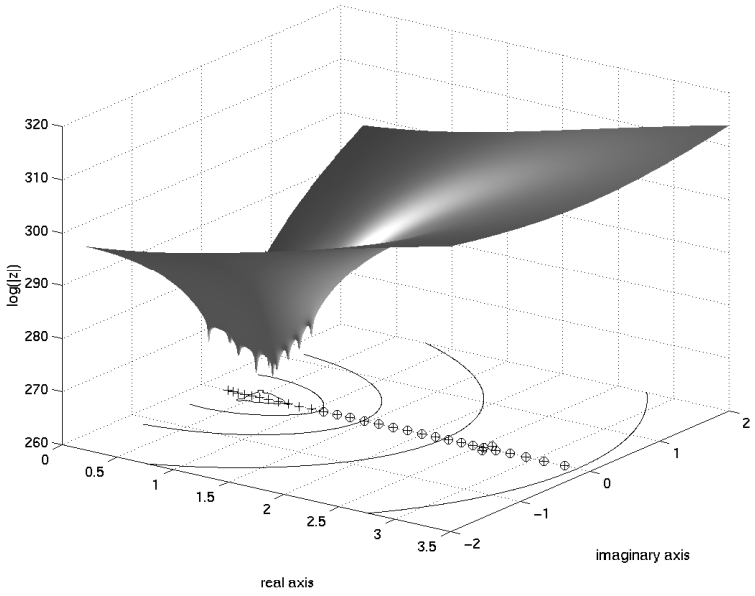

Figure 2.6 A 30-th degree filtering polynomial generated just before IRA converged.

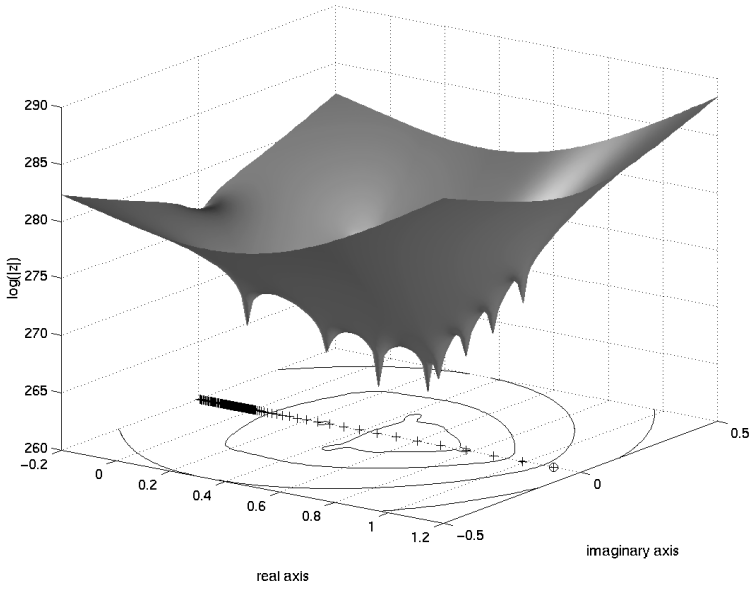

Figure 2.7 A different view of the filtering polynomial generated just before IRA converged.

Chapter 3

Rational Restarting and Truncated RQ-iteration

The Hessenberg structure of the projected matrix H_m and the top-to-bottom bulge chase strategy made it possible to implement a truncated version of the QR iteration within the Arnoldi factorization. As shown in Chapter 2, the QR update implicitly replaces the starting vector v_0 of an Arnoldi factorization with $\psi(A)v_0$ where $\psi(\lambda)$ is a polynomial designed to filter out the unwanted eigencomponents from v_0. The IRA algorithm has been successfully used in a number of applications [59, 35, 64, 83, 90]. However, there are situations where IRA converges slowly. One particular instance occurs in computational chemistry where the eigenvalues of interest are clustered around zero. The difficulty is that the spectrum contains many dominant but uninteresting eigenvalues. In this case, the polynomial constructed by IRA cannot completely eliminate the influence of the unwanted eigencomponents from the starting vector within a reasonable amount of time. In this section, we consider using a rational restarting strategy that repeatedly modifies the starting vector v_0 of the Arnoldi iteration by applying a rational function of A to v_0. The rational function has the form

$$\psi(\lambda) = \frac{1}{(\lambda - \mu_1)(\lambda - \mu_2)\cdots(\lambda - \mu_p)},$$

where $\mu_1, \mu_2, ..., \mu_p$ are shifts to be defined in Section 3.2.2. The implementation of a rational restart can be done implicitly via a truncated RQ iteration [84]. In Section 3.1, we briefly describe the RQ iteration, and derive a truncated version that is in the same spirit as IRA. The truncated RQ iteration requires solving a linear system with a special structure. This system is defined in Section 3.1. It is referred to as the *TRQ equation*. Some theoretical issues regarding the properties of the TRQ

equation are examined there. A mechanism for solving this equation is provided in Section 3.2.1. Other implementation details are discussed in Section 3.2.2–3.2.3. If the TRQ equation is solved with a direct method (based on Gauss elimination) the convergence of the TRQ iteration is swift. (It is quadratic in general and cubic for symmetric problems.) One would like to know what happens if the direct solver is replaced by a preconditioned iterative solver. The convergence of such an "inexact TRQ" scheme is analyzed in Section 3.4. It is shown that the convergence is locally linear under some appropriate assumptions.

3.1 Truncating an RQ iteration

The RQ iteration differs from the QR iteration only in the way a Hessenberg matrix is factored. Again, the iteration begins with a reduction to Hessenberg form

$$AV = VH, \tag{3.1}$$

and is followed by the actions described in Figure 3.1. Notice that the QR factorization employed in (2.2) is replaced by a RQ factorization.

Mathematically, a single *RQ sweep* can be explained as follows. After the initial reduction, one factors $H - \mu I$ into a product

$$H - \mu I = RQ,$$

where Q is unitary and R is upper triangular. After subtracting μV from (3.1) and replacing $H - \mu I$ with RQ, one has

$$(A - \mu I)V = V(RQ).$$

Postmultiplying the above equation by Q^H yields

$$\begin{aligned} (A - \mu I)(VQ^H) &= VR \qquad (3.2) \\ \text{or } (A - \mu I)(VQ^H) &= (VQ^H)(QR). \end{aligned}$$

Algorithm 1: Implicitly Shifted RQ-iteration

Input: (A, V, H) with $AV = VH$, $V^H V = I$, and H is upper Hessenberg.
Output: (V, H) such that $AV = VH, V^H V = I$ and H is upper triangular.

1. for $j = 1, 2, 3, ...$ until *converged*,
  1.1. Select a shift $\mu \leftarrow \mu_j$;
  1.2. Factor $H - \mu I = RQ$;
  1.3. $H \leftarrow QHQ^H$; $V \leftarrow VQ^H$;
2. end;

Figure 3.1 Implicitly Shifted RQ-iteration.

One can further deduce, after adding μVQ^H back to (3.3) and using the fact

$$QR + \mu I = Q(H - \mu I)Q^H + \mu I = QHQ^H,$$

that

$$A(VQ^H) = (VQ^H)(QHQ^H).$$

Following the convention used before, let us set $V_+ = VQ^H, H_+ = QHQ^H$ and denote the first column of V and V_+ by v_1 and v_1^+ respectively. It follows from (3.2) that

$$(A - \mu I)v_1^+ = v_1\rho_{1,1},$$

where $\rho_{1,1} = e_1^T Re_1$. This implies that the first column of V_+ is what one would have obtained by applying one step of inverse iteration to v_1 with the shift μ. This property is preserved in all subsequent RQ sweeps. Thus, one would expect very rapid convergence of the leading columns of V to an invariant subspace of A.

In the large-scale setting it is generally impossible to carry out the full iteration involving $n \times n$ orthogonal similarity transformations. It would be desirable to truncate this update procedure after k steps to maintain and update only the leading

portion of the factorizations occurring in this sequence. However, this cannot be done directly since the RQ update starts from the lower right corner of H, and works its way up towards the $(1,1)$ entry. To complete the last k steps of the update, one must determine the $k+1$-st column of both $V\tilde{Q}$ and $H\tilde{Q}$, where

$$\tilde{Q} = \begin{pmatrix} I_k & 0 \\ 0 & \hat{Q} \end{pmatrix}$$

is the product of Given's rotations used to drive the lower $(n-k)\times(n-k)$ portion of H to an upper triangular form. The determination of these intermediate vectors leads to solving a set of equations to be defined below.

To be precise, let us partition $V = (V_k, \hat{V})$ where V_k denotes the leading k columns of V and let

$$H = \begin{pmatrix} H_k & M \\ \beta_k e_1 e_k^T & \hat{H} \end{pmatrix}$$

be partitioned conformally so that

$$A(V_k, \hat{V}) = (V_k, \hat{V}) \begin{pmatrix} H_k & M \\ \beta_k e_1 e_k^T & \hat{H} \end{pmatrix}. \tag{3.3}$$

Let μ be some given shift. In a full RQ-iteration, we would begin factoring $H - \mu I$ from right to left using Givens method, say, to obtain

$$H - \mu I = \begin{pmatrix} H_k - \mu I_k & \hat{M} \\ \beta_k e_1 e_k^T & \hat{R} \end{pmatrix} \begin{pmatrix} I_k & 0 \\ 0 & \hat{Q} \end{pmatrix}$$

where $\hat{H} - \mu I_{n-k} = \hat{R}\hat{Q}$. It follows that

$$(A - \mu I)(V_k, \hat{V}\hat{Q}^H) = (V_k, \hat{V}) \begin{pmatrix} H_k - \mu I_k & \hat{M} \\ \beta_k e_1 e_k^T & \hat{R} \end{pmatrix}. \tag{3.4}$$

At this point, all one needs to know in order to complete the RQ factorization is the first columns of $\hat{V}\hat{Q}$, $\hat{M}$ and $\hat{R}$. (See Figure 3.2.) If these vectors can be computed

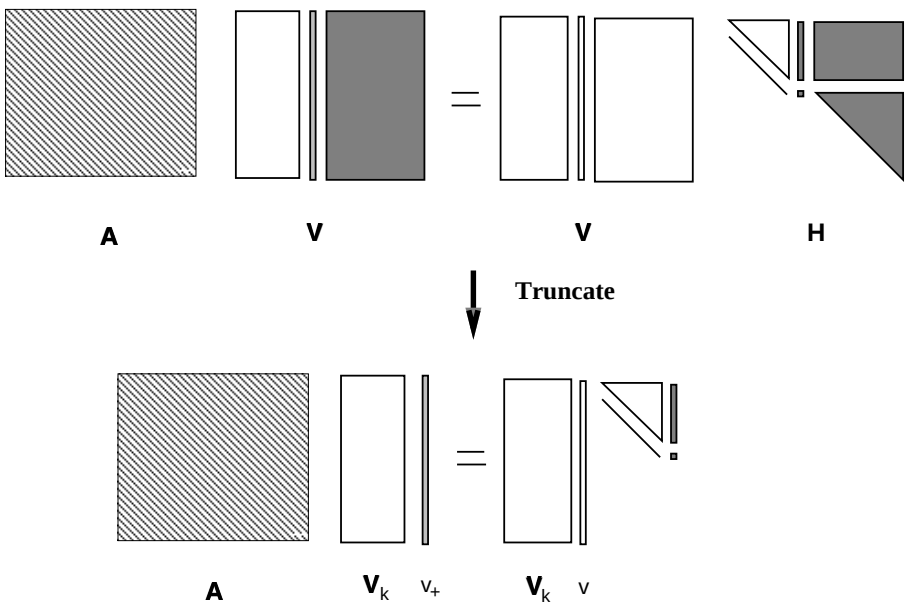

Figure 3.2 Truncating the RQ iteration.

without forming and applying $\hat{Q}$, then a truncated version of the RQ-iteration is possible. To determine these vectors, let us examine the first $k+1$ columns of (3.4). Let $v = \hat{V}e_1$, $v_+ = \hat{V}\hat{Q}^H e_1$, $h = \hat{M}e_1$, and $\alpha = e_1^T \hat{R} e_1$. It follows from (3.4) that

$$(A - \mu I)(V_k, v_+) = (V_k, v) \begin{pmatrix} H_k - \mu I_k & h \\ \beta_k e_k^T & \alpha \end{pmatrix}.$$

From this relationship, it follows that v_+ must satisfy

$$(A - \mu I)v_+ = V_k h + v\alpha, \tag{3.5}$$

with $V_k^H v_+ = 0$ and $\|v_+\| = 1$ since the columns of (V_k, v_+) must be orthonormal.

These conditions may be expressed succinctly through the *TRQ equations*

$$\begin{pmatrix} A - \mu I & V_k \\ V_k^H & 0 \end{pmatrix} \begin{pmatrix} v_+ \\ -h \end{pmatrix} = \begin{pmatrix} v\alpha \\ 0 \end{pmatrix}, \quad \|v_+\| = 1. \tag{3.6}$$

Note that the unknowns in (3.6) are v_+, h and α. The conditions $V_k^H v_+ = 0$ and $\|v_+\| = 1$ allow one to solve

$$\begin{pmatrix} A - \mu I & V_k \\ V_k^H & 0 \end{pmatrix} \begin{pmatrix} w \\ z \end{pmatrix} = \begin{pmatrix} v\gamma \\ 0 \end{pmatrix}$$

first for any $\gamma \neq 0$ and to normalize w to get v_+. Once we have v_+, premultiplying (3.5) with V_k and v yields

$$\begin{aligned} h &= V_k^H (A - \mu I) v_+ \\ \alpha &= v^H (A - \mu I) v_+ . \end{aligned}$$

Also, note that the first k columns on both sides of equation (3.4) satisfy a k-step Arnoldi relationship

$$(A - \mu I) V_k = V_k (H_k - \mu I_k) + f_k e_k^T \tag{3.7}$$

with $f_k = v\beta_k$. This relationship is important for analyzing the properties of (3.6) as we will show below.

The algorithm we shall develop depends upon the determination of v_+, h, and α directly from equation (3.6) rather than from the RQ factorization procedure. The fact that the RQ factorization exists assures us that a solution to (3.6) exists even when the bordered matrix in (3.6) is singular. The following series of lemmas drawn from [84] characterize how singularity can occur in these equations. They also establish that the solution to (3.6) is unique in any case. In the next section we show that the singular case in (3.6) is benign and easily dealt with numerically.

Lemma 3.1 Assume $A - \mu I$ is nonsingular (i.e., that μ is not an eigenvalue of A) and that equations (3.6) and (3.7) hold as a result of the partial RQ-factorization described by (3.4). Then the bordered matrix

$$B \equiv \begin{pmatrix} A - \mu I & V_k \\ V_k^H & 0 \end{pmatrix} \tag{3.8}$$

is nonsingular if and only if $V_k^H(A-\mu I)^{-1}V_k$ is nonsingular. Moreover, if $V_k^H(A-\mu I)^{-1}V_k$ is singular and z is any nonzero vector such that $V_k^H(A-\mu I)^{-1}V_k z = 0$, then $w = -(A-\mu I)^{-1}V_k z$ is nonzero, and $v_+ = \frac{w}{\|w\|}$, $h = -\frac{z}{\|w\|}$, and $\alpha = 0$ satisfy the TRQ equations.

Proof Since the RQ-factorization $\hat{R}\hat{Q} = \hat{H} - \mu I$ always exists, it follows that (3.6) must hold in any case. The assumption that $A - \mu I$ is nonsingular provides the block factorization

$$B = \begin{pmatrix} I & 0 \\ V_k^H(A-\mu I)^{-1} & I \end{pmatrix} \begin{pmatrix} A-\mu I & V_k \\ 0 & -V_k^H(A-\mu I)^{-1}V_k \end{pmatrix}. \tag{3.9}$$

Clearly, B is nonsingular if and only if $V_k^H(A-\mu I)^{-1}V_k$ is nonsingular.

To establish the second part of the lemma, we show that the equation

$$\begin{pmatrix} A-\mu I & V_k \\ 0 & V_k^H(A-\mu I)^{-1}V_k \end{pmatrix} \begin{pmatrix} w \\ z \end{pmatrix} = \begin{pmatrix} v \\ V_k^H(A-\mu I)^{-1}v \end{pmatrix} \alpha \tag{3.10}$$

has a nonzero solution $(w^H, z^H)^H$ with $\alpha = 0$ if and only if $V_k^H(A-\mu I)^{-1}V_k$ is singular.

To prove this suppose, first, that $\alpha = 0$ and $(w^H, z^H)^H$ is a nonzero solution to (3.10). Then $V_k^H(A-\mu I)^{-1}V_k$ must be singular because $A - \mu I$ is assumed to be nonsingular. On the other hand, if we assume $V_k^H(A-\mu I)^{-1}V_k$ is singular and z is a nonzero vector such that $V_k^H(A-\mu I)^{-1}V_k z = 0$, then putting $w = -(A-\mu I)^{-1}V_k z$ will provide a nonzero solution to (3.10) with $\alpha = 0$. Moreover, w must be nonzero since z is nonzero and $(A-\mu I)^{-1}V_k$ has linearly independent columns. Therefore, $v_+ = \frac{w}{\|w\|}$, $h = -\frac{z}{\|w\|}$, and $\alpha = 0$ will satisfy the TRQ equations. □

Lemma 3.1 indicates that the solution to (3.6) will be unique if and only if $V_k^H(A-\mu I)^{-1}V_k$ is either nonsingular or has a one-dimensional null space. The following lemma establishes this fact and hence the uniqueness of the solution to the TRQ equations (3.6).

Lemma 3.2 Assume $A-\mu I$ is nonsingular and that equations (3.6) and (3.7) hold as a result of the partial RQ-factorization described by (3.4). If $G \equiv V_k^H(A-\mu I)^{-1}V_k$ is singular, then the null space of G^H is $span\{e_k\}$.

Proof Let $y = V_k^H(A-\mu I)^{-1}f_k$ and define $H_\mu \equiv H_k - \mu I_k$. Then

$$\begin{aligned} GH_\mu &= V_k^H(A-\mu I)^{-1}V_k H_\mu \\ &= V_k^H(A-\mu I)^{-1}[(A-\mu I)V_k - f_k e_k^T] \\ &= I_k - y e_k^T. \end{aligned} \tag{3.11}$$

If G is singular, and x is any nonzero vector such that $0 = x^H G$, then (3.11) implies

$$0 = x^H G H_\mu = x^H - (x^H y)e_k^T.$$

Since $x \neq 0$ this equation implies $x^H y \neq 0$ which in turn implies that $x/(x^H y) = e_k$. Hence $e_k^T G = 0$ and the null space of G^H is $span\{e_k\}$. This concludes the proof. □

Finally, the following lemma indicates that exact singularity of B rarely occurs.

Lemma 3.3 Assume $A-\mu I$ is nonsingular and that equations (3.6) and (3.7) hold as a result of the partial RQ-factorization described by (3.4). Then $\alpha = 0$ in (3.6) and $V_k^H(A-\mu I)^{-1}V_k$ is singular if and only if the shift μ is an eigenvalue of $\hat{H}$ in equation (3.3).

Proof It is sufficient to show $V_k^H(A-\mu I)^{-1}V_k$ is singular if and only if the shift μ is an eigenvalue of $\hat{H}$ in equation (3.3). To this end, note that $V_k^H(A-\mu I)^{-1}V_k$ is singular if and only if $V_k^H(A-\mu I)^{-1}V_k z = 0$ for some $z \neq 0$. Since $(V_k, \hat{V})$ is unitary, any such z must satisfy

$$V_k z = (A-\mu I)\hat{V}g = (A-\mu I)(V_k, \hat{V}) \begin{pmatrix} 0 \\ g \end{pmatrix}$$

for some nonzero vector g (i.e. $(A-\mu I)^{-1}V_k z$ must be in the range of $\hat{V}$). This implies

$$\begin{aligned} V_k z &= (V_k, \hat{V}) \begin{pmatrix} H_k - \mu I_k & M \\ \beta_k e_1 e_k^T & \hat{H} - \mu I_{n-k} \end{pmatrix} \begin{pmatrix} 0 \\ g \end{pmatrix} \\ &= V_k M g + \hat{V}(\hat{H} - \mu I_{n-k}) g. \end{aligned}$$

Since $(V_k, \hat{V})$ is unitary, it follows that

$$(\hat{H} - \mu I_{n-k}) g = 0 \tag{3.12}$$

and since g is nonzero this implies the singularity of $(\hat{H} - \mu I_{n-k})$. Clearly this argument is reversible; i.e., if there is a nonzero g that satisfies (3.12) the reverse argument may be used to produce a nonzero z such that $V_k^H (A-\mu I)^{-1} V_k z = 0$ and the lemma is proved. □

The TRQ equations may be used to develop a truncated k-step version of the Implicitly Shifted RQ iteration. If a k-step Arnoldi factorization (3.7) has been obtained then a k-step TRQ iteration may be implemented as shown in Algorithm 2 (Figure 3.3).

The key idea here is to determine the $k+1$-st column v_+ of the updated matrix V and the $k+1$-st column of H that would have been produced in the RQ iteration by solving the linear system (3.6). Then, the iteration is completed through the normal RQ iteration. As eigenvalues converge, the standard deflation rules of the RQ iteration may be applied. Orthogonality of the basis vectors is explicitly maintained through the accurate solution of the TRQ equation. Moreover, even if the accuracy of this solution is relaxed, orthogonality may be enforced explicitly through the DGKS mechanism [14]. (Also see Section 2.1.) Potentially, the linear solve indicated at Step 2.2 of Algorithm 2 could be provided by a straightforward block elimination scheme.

Algorithm 2: (TRQ) Truncated RQ-iteration

Input: (A, V_k, H_k, f_k) with $AV_k = V_k H_k + f_k e_k^T$, $V_k^H V_k = I$, H_k is upper Hessenberg.
Output: (V_k, H_k) such that $AV_k = V_k H_k, V_k^H V_k = I$ and H_k is upper triangular.

1. Put $\beta_k = \|f_k\|$ and put $v = f_k/\beta_k$;
2. for $j = 1, 2, 3, \ldots$ until *convergence*,
  2.1. Select a shift $\mu \leftarrow \mu_j$;
  2.2. Solve $\begin{pmatrix} A-\mu I & V_k \\ V_k^H & 0 \end{pmatrix} \begin{pmatrix} v_+ \\ -h \end{pmatrix} = \begin{pmatrix} v\alpha \\ 0 \end{pmatrix}$ with $\|v_+\| = 1$;
  2.3. *RQ* Factor $\begin{pmatrix} H_k - \mu I_k & h \\ \beta_k e_k^T & \alpha \end{pmatrix} = \begin{pmatrix} R_k & r \\ 0 & \rho \end{pmatrix} \begin{pmatrix} Q_k & q \\ \sigma e_k^T & \gamma \end{pmatrix}$;
  2.4. $V_k \leftarrow V_k Q_k^H + v_+ q^H$;
  2.5. $\beta_k \leftarrow \sigma e_k^T R_k e_k$; $v \leftarrow v_k \bar{\sigma} + v_+ \bar{\gamma}$;
  2.6. $H_k \leftarrow Q_k R_k + \mu I_k$;
3. end;

Figure 3.3 The Truncated *RQ*-iteration.

However, considerable refinements to this scheme are possible due to the existing k-step Arnoldi relationship (3.7). This will be discussed in the next section.

3.2 Implementation issues in TRQ

In this section, we address some practicalities associated with an efficient implementation of the TRQ iteration.

3.2.1 Solving the TRQ Equations

The Truncated RQ iteration described in the previous section will only be effective in the large-scale setting if there is an efficient means for solving the TRQ equations. Recall that A, H_k, V_k and $f_k = v\beta_k$ are in a k-step Arnoldi relation (3.13) so that

$$(A - \mu I)V_k = V_k(H_k - \mu I_k) + f_k e_k^T. \tag{3.13}$$

Rescaling the right hand side of the system (3.6) by β/α leads to

$$\begin{pmatrix} A-\mu I & V_k \\ V_k^H & 0 \end{pmatrix} \begin{pmatrix} w \\ z \end{pmatrix} = \begin{pmatrix} f_k \\ 0 \end{pmatrix}. \tag{3.14}$$

Recall from the last section that rescaling (3.6) does not change the final solution since we can always recover v_+ through the normalization $v_+ = w/\|w\|$. If we put $d = (A-\mu I)^{-1} f_k$ and $y = V_k^H d$, then block Gaussian elimination leads to solving the equations

1. $V_k^H (A-\mu I)^{-1} V_k z = y$,
2. $(A-\mu I)w = f_k - V_k z$.

The first equation implies that one has to solve a block system of k equations to obtain $(A-\mu I)^{-1}V_k$. However, we will show in the following that this block solve is not necessary. Since V_k satisfies (3.13), one can take advantage of this special property of V_k to further simplify (3.14) and to derive a solution scheme that requires just a single linear solve with $A-\mu I$ as the coefficient matrix. Moreover, this efficient solution scheme does not depend on determining the singularity of the TRQ equations (3.6) in any way. The underlying theory is developed with the following lemma.

Lemma 3.4 Assume $A-\mu I$ is nonsingular and define $G \equiv V_k^H (A-\mu I)^{-1} V_k$ and $H_\mu \equiv (H_k - \mu I_k)$. There is a vector s such that either

$$(I_k - H_\mu G)s \neq 0 \quad \text{or} \quad e_k^T G s \neq 0. \tag{3.15}$$

For any such s, put

$$w \equiv (I - V_k V_k^H)(A-\mu I)^{-1} V_k s.$$

Then $w \neq 0$ and a solution v_+, h, α to (3.6) is given by

$$v_+ = w/\|w\|, \quad h = (I_k - H_\mu G)s/\|w\|, \quad \alpha = -\beta_k e_k^T G s/\|w\|.$$

Proof If $e_k^T Gs = 0$ for all vectors s, then the matrix $H_\mu G$ is singular and there must be a nonzero vector s such that $(I_k - H_\mu G)s \neq 0$. Therefore, there is a k-dimensional vector s that satisfies either $\theta \equiv e_k^T Gs \neq 0$ or $(I_k - H_\mu G)s \neq 0$.

For any such s, put $w \equiv (I - V_k V_k^H)(A - \mu I)^{-1} V_k s$. Observe that

$$\begin{aligned}(A - \mu I)w &= (A - \mu I)(I - V_k V_k^H)(A - \mu I)^{-1} V_k s \\ &= V_k s - (A - \mu I) V_k G s \\ &= V_k s - [V_k H_\mu + f_k e_k^T] G s \\ &= V_k (I_k - H_\mu G)s - f_k \theta. \end{aligned} \tag{3.16}$$

The conditions on s assure that the right hand side of (3.16) is nonzero. It follows that $w \neq 0$ and that

$$(A - \mu I)v_+ = V_k h + v\alpha$$

where $v_+ = w/\|w\|$, $h = (I_k - H_\mu G)s/\|w\|$, and $\alpha = -\beta_k \theta / \|w\|$. □

Remark 1 The original motivation for developing Lemma 3.4 was to handle the case when μ is an eigenvalue of H_k. A particular choice of s for this case is to put $s = q$ where $q^H H_\mu = 0$, and $q^H q = 1$. Then

$$q^H (I_k - H_\mu G)s = q^H s = q^H q = 1.$$

The conditions of Lemma 3.4 are clearly satisfied with this choice of s. However, we do not use this choice in practice.

Remark 2 The most general form of selecting a right hand side for constructing w is to take

$$w \equiv (I - V_k V_k^H)(A - \mu I)^{-1}(V_k t + f_k \eta)$$

where $s \equiv t - H_\mu e_k \eta$ is chosen to satisfy the conditions of Lemma 3.4. To see this, observe that

$$\begin{aligned} V_k t + f_k \eta &= V_k s + [V_k H_\mu + f_k e_k^T] e_k \eta \\ &= V_k s + (A - \mu I) V_k e_k \eta. \end{aligned}$$

Hence,

$$(I - V_k V_k^H)(A - \mu I)^{-1}(V_k t + f_k \eta) = (I - V_k V_k^H)(A - \mu I)^{-1} V_k s.$$

Thus, there is no mathematical reason to include the term $f_k \eta$, but the additional freedom may eventually have some numerical consequences that are not apparent at the moment. Note that when the shift μ is an eigenvalue of H_k then the combination of $t = 0$,$\eta = 1$ is prohibited because the corresponding vector s does not satisfy either of the conditions 3.15 required for constructing the solution in Lemma 3.4.

Remark 3 An alternative to forming h as described in Lemma 3.4 is to form w as described above and normalize to get $v_+ = w/\|w\|$. Then construct h and α using the DGKS mechanism [14] to orthogonalized the vector $(A - \mu I)v_+$ against V_k and f_k respectively. Thus

$$h \leftarrow V_k^H (A - \mu I) v_+ = V_k^H A v_+, \quad \alpha \leftarrow f_k^H (A - \mu I) v_+ / \| f_k \|.$$

The formulation just developed is appropriate when a sparse direct factorization of $A - \mu I$ is feasible. When this is not the case we must resort to an iterative scheme. For an iterative scheme, there may be an advantage to solving the projected equation

$$(I - V_k V_k^H)(A - \mu I)(I - V_k V_k^H)\hat{w} = f_k \tag{3.17}$$

Algorithm 3: Direct Solution of the TRQ Equations

Input: (A, V_k, H_k, f_k, μ) with $AV_k = V_k H_k + f_k e_k^T$, $V_k^H V_k = I$ and $V_k^H f_k = 0$.
Output: (v_+, h, α) such that $(A - \mu I)v_+ = V_k h + f_k \alpha$, $V_k^H v_+ = 0$ and $\|v_+\| = 1$.

1. Choose t and η and solve $(A - \mu I)w = V_k t + f_k \eta$;
2. $y \leftarrow V_k^H w$;
3. $w \leftarrow w - V_k y$;
4. $v_+ \leftarrow \frac{w}{\|w\|}$; $\alpha \leftarrow f_k^H (A - \mu I)v_+ / \|f_k\|$; $h \leftarrow V_k^H A v_+$;

Figure 3.4 Direct Solution of the TRQ Equations.

and putting

$$v_+ \leftarrow \frac{w}{\|w\|}$$

where $w = (I - V_k V_k^H)\hat{w}$. This is mathematically equivalent to solving the TRQ equations. The advantage here is that the matrix

$$(I - V_k V_k^H)(A - \mu I)(I - V_k V_k^H)$$

will likely be much better conditioned than $A - \mu I$ when μ is near an eigenvalue of A. A projected equation of this form plays a key role in the Jacobi-Davidson method recently developed in [79, 80, 21]. It also provides a means for allowing inaccurate solutions and preconditioning as we shall discuss later in Section 3.4.

3.2.2 Selection of Shifts

Another important issue regarding the TRQ iteration is the selection of shifts. Various options are available and they each lead to different convergence behavior. We discuss only a few of these options below.

The simplest strategy we have in mind is to use a fixed shift μ throughout the TRQ iteration. This shift is referred to as the *target* shift in the following discussion. In this case, a single matrix factorization of $A - \mu I$ may be used repeatedly to get inverse power method type of convergence. However, if the ratio

$$\sigma = \frac{|\lambda_j - \mu|}{|\lambda_{j+1} - \mu|} \tag{3.18}$$

is close to 1, the approximation to λ_j converges extremely slowly.

At the other extreme, we could adjust the shift at each iteration to enhance the rate of convergence. Ritz values (eigenvalues of H_k) that have not converged are natural candidates for the shift. They provide the best approximations to eigenvalues of A from the subspace spanned by the columns of V_k. Before each TRQ update we compute the Ritz values and identify those that have not converged as shift candidates. We then choose, from these candidates, a Ritz value nearest to the target as the next shift. This choice of shift is optimal in the sense that it often leads to a quadratic or cubic convergence rate. However, the rapid convergence is obtained at the cost of factoring a matrix at each iteration. It is observed from our experiments that Ritz values tend to jump around during the early stage of the TRQ iteration. Thus, the target shift may need to be reused during the first few iterations until the Ritz values settle down.

A reasonable compromise between the first and the second approach is to use a fixed shift until an eigenvalue has converged. Another possibility is to use each shift for (at most) a fixed number of iterations. In either case, the best Ritz value that has not yet converged may be selected as the next shift. Rapid convergence is generally obtained with this strategy. The cost for matrix factorization is reduced in comparison with the second approach. It is reported in [84] that this intermittent shifting scheme is very competitive with the Rational Krylov method of Ruhe [68, 67, 69].

Finally, the leading k-columns of the Implicitly Shifted RQ iteration may be obtained by selecting the same set of shifts as the dense algorithm if desired. For example, denoting the matrix elements of the upper triangle of H by γ_{ij}, we could use γ_{11} as the shift. This corresponds to the Rayleigh quotient shift in the RQ algorithm. Another alternative is the Wilkinson shift. This is defined to be the eigenvalue of the leading 2×2 matrix

$$\begin{pmatrix} \gamma_{11} & \gamma_{12} \\ \beta_1 & \gamma_{22} \end{pmatrix}$$

that is the nearest to γ_{11}. These strategies may be used when no target shift is given in advance, or when the order in which the eigenvalues are computed is not important.

Once the shift is chosen, an RQ update as described in Steps 2.3 through 2.6 of Algorithm 2 is taken. Clearly, it can be done explicitly, but there may be some advantage to an implicit application. An implicit shift application is straightforward since

$$\begin{pmatrix} H_k - \mu I_k & h \\ \beta_k e_k^T & \alpha \end{pmatrix} = \begin{pmatrix} H_k & h \\ \beta_k e_k^T & \tilde{\alpha} \end{pmatrix} - \mu \begin{pmatrix} I_k & 0 \\ 0 & 1 \end{pmatrix},$$

where $\tilde{\alpha} = \alpha + \mu$. Thus the standard bulge-chase implementation of an RQ sweep corresponding to the shift μ may be applied to the matrix

$$\begin{pmatrix} H_k & h \\ \beta_k e_k^T & \tilde{\alpha} \end{pmatrix}.$$

Finally, when the matrix A is real and non-symmetric, we would like to perform the TRQ iteration in real arithmetic. However, there seems to be no simple analog to the double shifting strategy used in the full RQ algorithm. Applying double shifts implicitly in the TRQ iteration is possible. However, the corresponding TRQ equation involves $\hat{A} = (A - \bar{\mu} I)(A - \mu I)$, and more work is required to solve this equation. We include the double shift RQ algorithm in the appendix. However, it is still questionable whether the double implicit shifting strategy should be practiced.

3.2.3 Deflation

As discussed earlier, in each TRQ iteration a defining equation

$$\begin{pmatrix} A-\mu I & V_k \\ V_k^H & 0 \end{pmatrix} \begin{pmatrix} v_+ \\ h \end{pmatrix} = \begin{pmatrix} v\alpha \\ 0 \end{pmatrix}$$

is solved so that a truncated Hessenberg reduction of the form

$$A(V_k, v_+) = (V_k, v) \begin{pmatrix} H_k & h \\ \beta_k e_k^T & \alpha \end{pmatrix} \tag{3.19}$$

is maintained. As the TRQ iteration proceeds, the leading sub-diagonal elements of H_k become small. When the magnitude of a sub-diagonal element β_j falls below some numerical threshold, the matrix H_k is split to give

$$H_k = \begin{pmatrix} H_j & M \\ 0 & \hat{H}_{k-j} \end{pmatrix}.$$

The first j columns of V_k form a basis for an invariant subspace of A, and j eigenvalues of A may be extracted from H_j. The deflation technique used in the QR algorithm can be applied here to obtain subsequent eigenvalues. We rewrite (3.19) as

$$(A-\mu I)(V_j, \hat{V}_{k-j}, v^+) = (V_j, \hat{V}_{k-j}, v) \begin{pmatrix} H_j - \mu I_j & M & h_1 \\ 0 & \hat{H}_{k-j} - \mu I_{k-j} & h_2 \\ 0 & \beta_k e_{k-j}^T & \alpha \end{pmatrix}, \tag{3.20}$$

where

$$V_k = (V_j, \hat{V}_{k-j}) \quad \text{and} \quad h = \begin{pmatrix} h_1 \\ h_2 \end{pmatrix},$$

have been partitioned conformally with V_j representing the leading j columns of V_k and h_1 representing the first j components of h.

An upper triangular matrix $\hat{R}$ and an orthogonal matrix $\hat{Q}$ of the form

$$\hat{R} = \begin{pmatrix} R_2 & r \\ 0 & \rho \end{pmatrix}, \quad \hat{Q} = \begin{pmatrix} Q_2 & q \\ \sigma e_{k-j}^T & \gamma \end{pmatrix}$$

are constructed such that

$$\begin{pmatrix} \hat{H}_{k-j} - \mu I_{k-j} & h_2 \\ \beta_k e_{k-j}^T & \alpha \end{pmatrix} = \hat{R}\hat{Q}.$$

Multiplying (3.20) from the right by $\tilde{Q}^H = \begin{pmatrix} I_j & \\ & \hat{Q}^H \end{pmatrix}$ yields

$$(A - \mu I)(V_j, \hat{V}_{k-j}^+, \hat{v}_+) = (V_j, \hat{V}_{k-j}, v) \begin{pmatrix} H_j - \mu I_j & \hat{M} & \hat{h}_1 \\ 0 & R_2 & r \\ 0 & 0 & \rho \end{pmatrix},$$

where $\hat{V}_{k-j}^+ = \hat{V}_{k-j} Q_2^H + v_+ q^H$, $\hat{v}_+ = \bar{\sigma}\hat{V}_{k-j} e_{k-j} + \bar{\gamma} v_+$, $\hat{M} = M Q_2^H + h_1 q^H$, and $\hat{h}_1 = \bar{\sigma} M e_{k-j} + \bar{\gamma} h_1$. Note that V_j and H_j are not modified during the deflation.

The next cycle of TRQ iteration starts with the selection of a new shift. The role of $\hat{H}_{k-j}$,$\hat{V}_{k-j}$ and v are replaced by $\hat{H}_{k-j}^+ = Q_2 R_2 + \mu I_{k-j}$, $\hat{V}_{k-j}^+$ and $\hat{v}_+$ respectively. If the sub-diagonal elements of H_k converge to zero in order (from top to bottom,) a partial Schur form

$$AV_j = V_j R_j,$$

is obtained. Of course, when a subdiagonal β_j approaches zero out of order, then the splitting described in equation (3.20) above will still yield a partial Schur form since the Schur form of $H_j Q_j = Q_j R_j$ can be used to make an explicit transformation.

3.3 Numerical Examples

A simple example is shown in this section to illustrate the convergence of the TRQ iteration. Recall that the rate of convergence of TRQ is exactly the same as that of

the full RQ iteration if the TRQ equations (3.6) are solved exactly. In this example, we let the matrix A be a 100×100 standard 5-point discrete Laplacian defined on $[0,1] \times [0,1]$ with zero Dirichlet boundary condition. For simplicity, the matrix is scaled by h^2, where $h = 1/101$ is the mesh size of the discretization. We seek 4 eigenvalues nearest to zero, hence always choose the unconverged Ritz value with the smallest magnitude as the shift. We set the size of the Arnoldi factorization to be 5 ($k = 5$), and maintain orthogonality between the Arnoldi vectors using the DGKS scheme [14]. Table 3.1 lists the sub-diagonal element β_j ($j = 1, 2, 3, 4$) of H_5 at each iteration. Once $|\beta_j|/(|H_{j,j}| + |H_{j+1,j+1}|)$ drops below a prescribed tolerance of 10^{-15}, we set β_j to zero and "lock" the first j columns of H_5 and V_5 as illustrated in equation (3.20). Clearly, the first eigenvalue converges cubically, and the second one shows cubic convergence rate after the first one has converged. At the end of the 12-th iteration, all four eigenvalues

$$\lambda_1 = 0.16203$$

$$\lambda_2 = 0.39851$$

$$\lambda_3 = 0.39851$$

$$\lambda_4 = 0.63499$$

have been found: The convergence criterion here used a tolerance of 10^{-15} in the test for declaring a sub-diagonal element to be zero. The computed direct residuals for all converged eigenpairs were on the order of 10^{-15}. The multiplicity of the eigenvalue 0.39851 is detected.

3.4 Inexact TRQ

If the cost of factoring $A - \mu I$ is moderate, the rational restart of an Arnoldi factorization via the TRQ update provides a clean and efficient way of obtaining accurate

iteration	μ	β_1	β_2	β_3	β_4
1	0.18638	2.31×10^{-2}	2.18×10^{0}	1.91×10^{0}	1.58×10^{0}
2	0.16204	2.33×10^{-7}	6.23×10^{-1}	1.84×10^{0}	2.18×10^{0}
3	0.16203	1.11×10^{-21}	2.10×10^{-1}	1.36×10^{0}	1.84×10^{0}
4	0.44417	0	7.92×10^{-2}	1.27×10^{-1}	1.55×10^{0}
5	0.39857	0	1.36×10^{-5}	3.83×10^{-2}	7.24×10^{-1}
6	0.39851	0	4.08×10^{-17}	1.36×10^{-1}	9.47×10^{-2}
7	0.40410	0	0	1.34×10^{-2}	3.14×10^{-2}
8	0.39851	0	0	3.84×10^{-8}	4.24×10^{-2}
9	0.39851	0	0	8.58×10^{-21}	5.71×10^{-2}
10	0.63614	0	0	0	2.15×10^{-3}
11	0.63499	0	0	0	1.52×10^{-10}
12	0.63499	0	0	0	1.88×10^{-28}

Table 3.1 Convergence history of the 4 computed eigenvalues of a 2-D Laplacian.

approximations to interior and/or clustered eigenvalues. Otherwise, we must resort to other means of solving the TRQ equation (3.14) or (3.17). A preconditioned iterative solver is a natural candidate. In the following, we will present an algorithm based on the idea of incorporating an iterative solver in the TRQ iteration, and analyze the convergence of this method.

3.4.1 The Algorithm

We shall ask the question of whether the TRQ iteration will still provide accurate eigenvalue approximations if the accuracy of the solution to the TRQ equation is relaxed. If convergence does occur in this *inexact* scheme, how fast can it be? To address these questions, let us first examine the consequence of replacing the exact solution v_+ of (3.14) with some approximation $\tilde{v}_+$.

The vector $\tilde{v}_+$ can be computed by applying an iterative solver to

$$(A - \mu I)w = V_k s, \tag{3.21}$$

for some $s \neq 0$, followed by

$$\tilde{v}_+ \leftarrow (I - V_k V_k^H)w, \quad \tilde{v}_+ \leftarrow \frac{\tilde{v}_+}{\|\tilde{v}_+\|}, \tag{3.22}$$

as suggested by Lemma 3.4. We may also apply the solver directly to the projected equation

$$(I - V_k V_k^H)(A - \mu I)(I - V_k V_k^H)w = v. \tag{3.23}$$

When μ is near an eigenvalue of A, the linear system (3.21) is extremely ill-conditioned. Solving it by an iterative method may be challenging. On the other hand, although the projected system (3.23) is generally better conditioned, the matrix vector multiplication involved is more costly, and the construction of a preconditioner for the projected matrix $(I - V_k V_k^H)(A - \mu I)(I - V_k V_k^H)$ is not straightforward.

In any case, one must provide an approximation $\tilde{v}_+$ such that

$$V_k^H \tilde{v}_+ = 0 \quad \text{and} \quad \|\tilde{v}_+\| = 1$$

are satisfied. To continue the TRQ iteration, we shall compute h and α such that

$$(A - \mu I)\tilde{v}_+ = V_k h + v\alpha \tag{3.24}$$

holds. However, the following lemma indicates that it is generally difficult to find a perfect match for (3.24).

Lemma 3.5 Suppose we solve (3.21) by a Krylov subspace method with a zero starting vector and no preconditioner to obtain an approximation w. If $\tilde{v}_+ \equiv (I - V_k V_k^H)w \neq 0$, then

$$(A - \mu I)\tilde{v}_+ \notin \text{span}\{V_k, v\}.$$

Proof Recall that V_k and v are generated by an Arnoldi process. Thus, if we let v_1 be the first column of V_k, then

$$V_k = \text{span}\{v_1, Av_1, A^2v_1, ..., A^{k-1}v_1\} \quad \text{and} \quad v \in \text{span}\{v_1, Av_1, A^2v_1, ..., A^kv_1\}.$$

It follows that

$$V_k s = p(A)v_1, \tag{3.25}$$

for some polynomial $p(\lambda)$ of degree at most $k-1$. Applying a Krylov subspace solver to (3.21) yields an approximate solution

$$w = q(A)V_k s,$$

where $q(\lambda)$ is another polynomial associated with the Krylov linear solver. It follows from (3.25) that $w = q(A)p(A)v_1$. If we put $\psi(\lambda) = q(\lambda)p(\lambda)$, it follows from our assumption that the degree of $\psi(\lambda)$ must be at least k for otherwise $\psi(A)v_1 \in \text{span}\{V_k\}$, and $\tilde{v}_+ = (I - V_k V_k^H)w = 0$. Now, let $z = V_k^H(A - \mu I)w$. Since V_k spans a k dimensional Krylov subspace associated with A and v_1, the vector $V_k z$ can be expressed as

$$V_k z = r(A)v_1,$$

for some polynomial $r(\lambda)$ of degree at most $k-1$. Hence,

$$\begin{aligned}
(A - \mu I)\tilde{v}_+ &= (A - \mu I)(I - V_k V_k^H)w \\
&= (A - \mu I)w - (A - \mu I)V_k z \\
&= (A - \mu I)\psi(A)v_1 - (A - \mu I)r(A)v_1 \\
&= \phi(A)v_1,
\end{aligned}$$

where $\phi(\lambda) = (\lambda - \mu)\Big[\psi(\lambda) - r(\lambda)\Big]$, a polynomial of degree of at least $k+1$. Therefore, we conclude that

$$\tilde{v}_+ \notin \text{span}\{v_1, Av_1, ..., A^kv_1\} = \text{span}\{V_k, v\}.$$

□

Consequently, the best one can hope for from (3.24) is a weak solution derived from

$$V_k^H\Big((A-\mu I)\tilde{v}_+ - V_k\tilde{h} - v\tilde{\alpha}\Big) = 0 \tag{3.26}$$

$$v^H\Big((A-\mu I)\tilde{v}_+ - V_k\tilde{h} - v\tilde{\alpha}\Big) = 0 \tag{3.27}$$

or equivalently,

$$\tilde{h} = V_k^H A\tilde{v}_+, \quad \text{and} \quad \tilde{\alpha} = v^H(A-\mu I)\tilde{v}_+.$$

Due to the error remaining in (3.24), the truncated Hessenberg reduction (3.19) is now inexact. We can express this inexact reduction by

$$(A-\mu I)(V_k,\tilde{v}_+) = (V_k,v)\begin{pmatrix} H_k-\mu I_k & \tilde{h} \\ \beta_k e_k^T & \tilde{\alpha} \end{pmatrix} + ze_{k+1}^T, \tag{3.28}$$

where z is the residual error defined as

$$z \equiv (A-\mu I)\tilde{v}_+ - (V_k,v)\begin{pmatrix} \tilde{h} \\ \tilde{\alpha} \end{pmatrix}.$$

Recall from (3.26) and (3.27) that $V_k^H z = 0$ and $v^H z = 0$. If we now proceed by applying a sequence of Given's rotations from the right to (3.28) to annihilate the sub-diagonal elements of

$$\begin{pmatrix} H_k-\mu I_k & \tilde{h} \\ \beta_k e_k^T & \tilde{\alpha} \end{pmatrix},$$

the residual error will be mixed into all columns of V_k. Consequently, the updated basis vectors are no longer valid Arnoldi vectors. However, as we will show in the next section, the first column v_1^+ of this updated basis satisfies

$$(A-\mu I)v_1^+ = \rho_{11}v_1 + z\delta,$$

where δ is a product of sines associated with the aforementioned Given's rotations. This observation reveals that an approximate inverse iteration remains in this inexact

TRQ update. The error associated with this inverse iteration is likely to be considerably smaller than $\|z\|$ due the factor δ. Therefore, a simple remedy for correcting the contaminated Arnoldi basis is to recompute an Arnoldi factorization from the very first column of the updated V_k. An algorithm based on the above discussion is given in Figure 3.5.

Algorithm 4: (ITRQ) Inexact TRQ iteration

Input: (A, V_k, H_k, f_k) with $AV_k = V_k H_k + f_k e_k^T, V_k^H V_k = I$,
H_k is upper Hessenberg.
Output: (V_k, H_k) such that $AV_k = VH_k, V_k^H V_k = I$ and H_k is upper triangular.

1. Put $\beta_k = \|f_k\|$ and put $v = f_k/\beta_k$;
2. for $j = 1, 2, 3, ...$ until *converged*,
2.1. Select a shift $\mu \leftarrow \mu_j$;
2.2. Solve $(I - V_k V_k^H)(A - \mu I)(I - V_k V_k^H)w = v$ approximately;
2.3. $w \leftarrow (I - V_k V_k^H)w$, $v_+ \leftarrow w/\|w\|$;
2.4. $h \leftarrow V_k^H A v_+, \quad \alpha \leftarrow v^H (A - \mu I) v_+$;
2.5. *RQ* Factor $\begin{pmatrix} H_k - \mu I_k & h \\ \beta_k e_k^T & \alpha \end{pmatrix} = \begin{pmatrix} R_k & r \\ 0 & \rho \end{pmatrix} \begin{pmatrix} Q_k & q \\ \sigma e_k^T & \gamma \end{pmatrix}$;
2.6. $v_1 \leftarrow V_k Q_k^H e_1 + v_+ q^H e_1$;
2.7. $(H_k, V_k, v, \beta_k) \leftarrow$ Arnoldi(A, v_1, k);
3. end;

Figure 3.5 Inexact TRQ iteration.

3.4.2 Convergence Analysis

This section focuses on analyzing the convergence of this inexact TRQ scheme. In particular, we are interested in understanding the tradeoff between the accuracy of the solution to the TRQ equation and the rate of convergence in TRQ. For a simple

case in which Rayleigh quotient shifts are used throughout the TRQ iteration, we establish the linear convergence of the first Arnoldi basis vector to an eigenvector of A. The convergence factor depends on $\|\delta z\|$, the magnitude of the damped residual error in (3.28), and the gap between two consecutive eigenvalues sought.

To begin the analysis, let us assume that an inexact Hessenberg reduction (3.28) has been obtained, and $k-1$ rotations Q_1^H, Q_2^H,...,Q_{k-1}^H, each of the form

$$Q_i^H = \begin{pmatrix} I_{k-i} & & & 0 \\ & \gamma_i & \sigma_i & \\ & -\sigma_i & \gamma_i & \\ 0 & & & I_{i-1} \end{pmatrix}, \quad \sigma_i^2 + \gamma_i^2 = 1$$

have been applied to (3.28) from the right to annihilate the sub-diagonal elements of

$$\begin{pmatrix} H_k - \mu I_k & \tilde{h} \\ \beta_k e_k^T & \tilde{\alpha} \end{pmatrix}.$$

The first two columns of the new matrix equation satisfy

$$(A - \mu I)(v_1, \tilde{v}_2) = (v_1, v_2) \begin{pmatrix} 0 & \eta \\ \epsilon & \rho \end{pmatrix} + (0, z\hat{\sigma}), \tag{3.29}$$

where $\tilde{v}_2 = (V_k, \tilde{v}_+) Q_1^H Q_2^H \cdots Q_{k-1}^H e_2$, $\hat{\sigma} = \sigma_1 \sigma_2 \cdots \sigma_{k-1}$ and $\epsilon = \|(A - \mu I)v_1\|$. Now, let

$$\tau = \sqrt{\epsilon^2 + \rho^2}, \quad \sigma_k = \frac{\rho}{\tau}, \quad \text{and} \quad \gamma_k = \frac{\epsilon}{\tau}.$$

Applying $\begin{pmatrix} \gamma_k & \sigma_k \\ -\sigma_k & \gamma_k \end{pmatrix}$ to (3.29) from the right yields

$$(A - \mu I)(v_1^+, \tilde{v}_2^+) = (v_1, v_2) \begin{pmatrix} -\sigma_k \eta & \gamma_k \eta \\ 0 & \tau \end{pmatrix} + (-\sigma_k \hat{\sigma} z, \gamma_k \hat{\sigma} z), \tag{3.30}$$

where $v_1^+ = \gamma_k v_1 - \sigma_k \tilde{v}_2$ and $\tilde{v}_2^+ = \sigma_k v_1 + \gamma_k \tilde{v}_2$.

The convergence of the inexact TRQ iteration will be analyzed by examining the norm of $r_+ = (A - \mu_+ I)v_1^+$, where $\mu_+ = (v_1^+)^H A v_1^+$. We define the damped residual error ν as

$$\nu = \|\hat{\sigma} z\|. \tag{3.31}$$

The following lemma asserts that if the inexact TRQ is converging to an isolated eigenvalue of A, r_+ must satisfy $\|r_+\| \leq \psi(\mu, \nu)\|r\|$, where $\psi(\mu, z)$ is uniformly bounded if μ is sufficiently close to an isolated eigenvalue of A, and if ν is not too large.

Lemma 3.6 Let $r = (A - \mu I)v_1$ and $r_+ = (A - \mu_+)v_1^+$, where v_1 and v_1^+ are as defined in (3.30), and μ, μ_+ are Rayleigh quotients of A with respect to v_1 and v_1^+, respectively. If $A - \mu I$ is nonsingular, and μ is convergent to a simple eigenvalue of A. Then

$$\|r_+\| \leq \psi(\mu, \nu)\|r\|, \tag{3.32}$$

where the magnitude of the function ψ depends on μ and the size of the damped error ν defined in (3.31). Let $V \equiv (V_k, \hat{V}_{n-k})$ be unitary, where V_k consists of Arnoldi basis vectors generated by Step 2.7 in Algorithm 4. Repartition V as $V = (v_1, \hat{V}_{n-1})$. Let

$$C = \hat{V}_{n-1}^H A V_{n-1}, \quad \text{and} \quad \zeta = \|(C - \mu I)^{-1}\|^{-1}.$$

If $\nu < \zeta$, then

$$|\psi(\mu, z)| \leq \frac{|\epsilon\eta|}{\zeta^2 - \nu^2} + \frac{|\epsilon|\nu}{\zeta^2 - \nu^2} + \frac{\nu}{\sqrt{\zeta^2 - \nu^2}}, \tag{3.33}$$

where $\epsilon = \|(A - \mu)v_1\|$ and $\eta = v_1^H A v_2$.

Proof For clarity, we drop the subscripts of σ_k and γ_k in the following. Note that

$$\begin{aligned} r_+ &= (A-\mu_+ I)v_1^+ \\ &= (A-\mu I)v_1^+ + (\mu-\mu_+)v_1^+ \\ &= (-\sigma\eta)v_1 + (\mu-\mu_+)v_1^+ + (-z\hat{\sigma})\sigma. \end{aligned} \tag{3.34}$$

The last step of the derivation used the relation

$$(A-\mu I)v_1^+ = v_1(-\sigma_k\eta) - \hat{\sigma}\sigma_k z,$$

which appears in the first column of (3.30). The distance between μ_+ and μ may be estimated as follows:

$$\begin{aligned} \mu_+ - \mu &= (v_1^+)^H A v_1^+ - \mu \\ &= (v_1^+)^H (A-\mu I)v_1^+ \\ &= (v_1^+)^H(-\sigma\eta v_1 - z\hat{\sigma}\sigma) \\ &= (\gamma v_1 + \sigma\tilde{v}_2)^H(-\sigma\eta v_1 - z\hat{\sigma}\sigma) \\ &= -\gamma\sigma\eta - \sigma^2\hat{\sigma}\tilde{v}_2^H z. \end{aligned} \tag{3.35}$$

The last equality follows from the fact that $v_1^H z = 0$ and $v_1^H\tilde{v}_2 = 0$.

We will transform r_+ to $V^H r_+$ before checking its norm. (Since $V^H V = I$, $\|r_+\| = \|V^H r_+\|$.) Recall that $V = (v_1, \hat{V}_{n-1})$. Put $p = \hat{V}_{n-1}^H\tilde{v}_2$ and $\hat{z} = \hat{V}_{n-1}^H z$. We will need the following formulae to simplify the expression for $V^H r_+$:

$$V^H v_1^+ = V^H(\gamma v_1 - \sigma\tilde{v}_2) = \begin{pmatrix} \gamma \\ -\sigma p \end{pmatrix}, \quad \text{and} \quad V^H z = \begin{pmatrix} 0 \\ \hat{z} \end{pmatrix}.$$

Since $V_k^H z = 0$ and $v^H z = 0$, the first k components of $\hat{z}$ are zeros. Clearly, $\|\hat{z}\| = \|z\|$. Now, it follows from (3.34) and (3.35) that

$$V^H r_+ = V^H(A-\mu_+ I)v_1^+$$

$$
\begin{aligned}
&= (-\sigma\eta)V^H v_1 + (\mu-\mu_+)V^H v_1^+ - (\hat{\sigma}\sigma)V^H z \\
&= (-\sigma\eta)e_1 + \left[\gamma\sigma\eta + \sigma^2(\tilde{v}_2^H z)\hat{\sigma}\right]\begin{pmatrix}\gamma\\ -\sigma p\end{pmatrix} - \sigma\begin{pmatrix}0\\ \hat{z}\hat{\sigma}\end{pmatrix} \\
&= (-\sigma\eta)\begin{pmatrix}1-\gamma^2\\ \gamma\sigma p\end{pmatrix} + \sigma^2(\tilde{v}_2^H z)\hat{\sigma}\begin{pmatrix}\gamma\\ -\sigma p\end{pmatrix} - \sigma\begin{pmatrix}0\\ \hat{z}\hat{\sigma}\end{pmatrix} \\
&= (-\sigma\eta)\begin{pmatrix}\sigma^2\\ \gamma\sigma p\end{pmatrix} + \sigma^2(\tilde{v}_2^H z)\hat{\sigma}\begin{pmatrix}\gamma\\ -\sigma p\end{pmatrix} - \sigma\begin{pmatrix}0\\ \hat{z}\hat{\sigma}\end{pmatrix}.
\end{aligned}
$$

It is easy to verify that $\|p\| = 1$ since $p = \hat{V}_{n-1}^H \tilde{v}_2$, $\hat{V}_{n-1}^H \hat{V}_{n-1} = I_{n-1}$ and $\tilde{v}_2^H \tilde{v}_2 = 1$. Thus,

$$
\begin{aligned}
\|r_+\| = \|V^H r_+\| &= \|(-\sigma\eta)\begin{pmatrix}\sigma^2\\ \gamma\sigma p\end{pmatrix} + \sigma^2(\tilde{v}_2^H z)\hat{\sigma}\begin{pmatrix}\gamma\\ -\sigma p\end{pmatrix} - \sigma\begin{pmatrix}0\\ \hat{z}\hat{\sigma}\end{pmatrix}\| \\
&\le \sigma^2|\eta| + \sigma^2\|z\hat{\sigma}\| + \sigma\|z\hat{\sigma}\| \qquad (3.36)\\
&= \sigma^2|\eta| + \sigma^2\nu + \sigma\nu. \qquad (3.37)
\end{aligned}
$$

Recall that σ is generated to annihilate the sub-diagonal element of

$$
\begin{pmatrix}0 & \eta\\ \epsilon & \rho\end{pmatrix},
$$

which appears in (3.29). Now, since

$$
|\sigma| = \frac{|\epsilon|}{\sqrt{\epsilon^2+\rho^2}} \le |\frac{\epsilon}{\rho}| \quad \text{and} \quad |\epsilon| = \|r\|,
$$

we conclude that

$$
\|r_+\| \le \psi(\mu, z)\|r\|,
$$

where

$$
\psi(\mu, z) = \frac{|\epsilon\eta|}{\rho^2} + \frac{|\epsilon|\nu}{\rho^2} + \frac{\nu}{|\rho|}.
$$

Clearly, the factor $\psi(\mu, z)$ can be bounded uniformly if ν is not too large, and if $|\rho|$ can be bounded away from zero. Of course, one would not know the size of ρ until $k-1$ rotations $Q_1, Q_2, ..., Q_{k-1}$ have been applied. The following arguments provide an *a prior* lower bound for $|\rho|$. It asserts that $|\rho|$ can be bounded from below if ν is sufficiently small.

Recall from (3.30) that

$$(A - \mu I)v_1^+ = v_1(-\sigma\eta) + (-\sigma z\hat{\sigma}).$$

This is equivalent to

$$V^H(A - \mu I)VV^H v_1^+ = V^H v_1(-\sigma\eta) - \sigma \begin{pmatrix} 0 \\ \hat{z}\hat{\sigma} \end{pmatrix},$$

or

$$\begin{pmatrix} 0 & h^H \\ \epsilon e_1 & C - \mu I \end{pmatrix} \begin{pmatrix} \gamma \\ -\sigma p \end{pmatrix} = \begin{pmatrix} -\sigma\eta \\ -\sigma\hat{\sigma}\hat{z} \end{pmatrix},$$

where $h = v_1^H A\hat{V}_{n-1}$. It follows that

$$h^H p = \eta, \quad \text{and} \tag{3.38}$$

$$\rho e_1 - (C - \mu I)p = -\hat{\sigma}\hat{z}. \tag{3.39}$$

Since $e_1^T\hat{z} = 0$, it follows from (3.39) that

$$(C - \mu I)p = \begin{pmatrix} \rho \\ \hat{\sigma}\breve{z} \end{pmatrix},$$

where $\breve{z}$ denotes the vector consisting of the last $n-2$ components of z. The assumption that μ is convergent to a simple eigenvalue of A ensures that $C - \mu I$ is nonsingular. Thus

$$p = (C - \mu I)^{-1} \begin{pmatrix} p \\ \hat{\sigma}\breve{z} \end{pmatrix}.$$

Recall that $p = \hat{V}_{n-1}\tilde{v}_2$ has unit length. Therefore,

$$1 = \|p\| \leq \|(C - \mu I)^{-1}\| \sqrt{\rho^2 + \hat{\sigma}^2 \|z\|^2}.$$

or,

$$\frac{1}{\|(C - \mu I)^{-1}\|} \leq \sqrt{\rho^2 + \nu^2}. \tag{3.40}$$

Clearly, to establish a lower bound on ρ, one must prevent ν from getting too large.

Let $\zeta = 1/\|(C - \mu)^{-1}\|$. It follows from the assumption that $\nu < \zeta$ and equation (3.40) that

$$\zeta^2 - \nu^2 \leq \rho^2, \quad \text{or} \quad |\rho| \geq \sqrt{\zeta^2 - \nu^2}.$$

Consequently, we have

$$\psi(\mu, z) \leq \frac{|\epsilon\eta|}{\zeta^2 - \nu^2} + \frac{|\epsilon|\nu}{\zeta^2 - \nu^2} + \frac{\nu}{\sqrt{\zeta^2 - \nu^2}}. \tag{3.41}$$

□

Remark 1 As μ becomes sufficiently close to the desired eigenvalue, we may ignore the effect of the first two terms of (3.41), and focus on the dominating third term. It easy to verify that if

$$\nu < \frac{\zeta}{\sqrt{2}},$$

$|\psi(\mu, z)|$ can be strictly bounded by 1. Monotonic convergence can be expected in this case.

Remark 2 The above analysis is valid when the TRQ equation is solved exactly. One recovers the quadratic (cubic if A is Hermitian) convergence rate of the Rayleigh quotient iteration. To see this, we replace equations (3.34) and (3.35) with

$$\begin{aligned} r_+ &= (-\sigma\eta)v_1, \quad \text{and} \\ \mu_+ - \mu &= -\gamma\sigma\eta \end{aligned}$$

and conclude from (3.37) that

$$\|r_+\| = \|V^H r_+\| = \sigma^2 |\eta|.$$

Since $\sigma = \epsilon/(\epsilon^2 + \rho^2)$ and $\epsilon = \|r\|$,

$$\|r_+\| = \frac{|\eta|}{\sqrt{\epsilon^2 + \rho^2}} \|r\|^2 \leq \left|\frac{\eta}{\rho}\right| \|r\|^2. \tag{3.42}$$

It follows from (3.39) that $\rho e_1 = (C - \mu I)p$. Thus $|\rho|$ is bounded below by $1/\|(C - \mu I)^{-1}\|$, and quadratic convergence follows from (3.42). When A is Hermitian, $|\eta| = |h^H p| \leq \|h\|\|p\| = |\epsilon| = \|r\|$, and the cubic convergence rate follows.

Remark 3 We should point out that the bound given by (3.41) is not tight. This is a consequence of using the triangular inequality in (3.36). Thus, in practice, the requirement $\|z\| \leq \zeta$ may be relaxed.

Remark 4 For Hermitian problems, ζ is approximately the gap between the eigenvalue to which the inexact TRQ is converging to and the eigenvalue nearest it. This can often be estimated by examining $|\mu - \hat{\mu}|$, where $\hat{\mu}$ is the eigenvalue of H_k that is nearest to μ.

3.4.3 Numerical Examples

All computations shown in this section are performed in MATLAB on a SUN-Ultra2. Two iterative solvers MINRES [55] and GMRES [76] are used in the following examples. Both solvers construct approximate solutions to a linear system from a Krylov subspace. The MINRES algorithm is mainly used for solving symmetric indefinite systems. Since it can be implemented by a short recurrence, only a few vectors need to be stored. A k-step GMRES algorithm requires an orthogonal basis of a

k-dimensional Krylov subspace to be saved. To reduce the storage cost, the GMRES algorithm is often restarted using the approximate solution obtained in the previous run as a starting guess. We will use the notation `GMRES(k,m)` to denote a k-step GMRES with a maximum of m restarts. We set the convergence tolerance in both solvers to be 10^{-8}. We will also use ITRQ(k,m) to denote an inexact TRQ iteration in which k eigenpairs are to be computed and an m-step Arnoldi factorization is maintained.

Example 1 - Linear Convergence

The validity of the analysis presented in Section 3.4.2 is verified by a simple numerical example here. Again, we choose the familiar 100×100 tridiagonal matrix with 2 on the diagonal and -1 on the super and sub-diagonals as the test matrix. The gap between the first two smallest eigenvalues is $\xi = 2.9 \times 10^{-3}$. To show the local convergence rate, we choose the starting vector v_0 of the initial Arnoldi factorization to be

$$v_0 = z_1 + 0.01 \cdot r,$$

where z_1 is the eigenvector corresponding to the smallest eigenvalue (λ_1) of A, and r is a normally distributed random vector. We apply `GMRES(10,5)` to (3.23) to obtain the vector $\tilde{v}_+$ used in (3.24). The residual error associated with the equation (3.24) and the first sub-diagonal element β_1 of the tridiagonal matrix are displayed in Table 3.2. The $(1,1)$-entry of the matrix, denoted by α_1, is also listed. As ITRQ converges, we expect to see $\alpha \to \lambda_1$, $v_1 \to z_1$, and $\beta_1 = \|Av_1 - \alpha_1 v_1\| \to 0$.

We observe from column 4 that β_1 decreases monotonically in a linear fashion. This is in agreement with the theory developed in Section 3.4.2 since the residual error of (3.24) is less than the distance between the first two eigenvalues of A.

iteration	α_1	$\|z\|$	β_1
1	3.0244×10^{-3}	-	7.8×10^{-3}
2	9.6750×10^{-4}	1.9×10^{-3}	7.3×10^{-5}
3	9.6742×10^{-4}	2.5×10^{-3}	3.2×10^{-6}
4	9.6744×10^{-4}	5.7×10^{-4}	1.5×10^{-7}
5	9.6744×10^{-4}	1.6×10^{-3}	6.7×10^{-9}
6	9.6744×10^{-4}	9.2×10^{-4}	3.2×10^{-10}
7	9.6744×10^{-4}	1.3×10^{-3}	1.6×10^{-11}

Table 3.2 The convergence of inexact TRQ.

Example 2 - Compute Several Eigenvalues

The following example illustrates that one can use the inexact TRQ iteration to compute more than one eigenpair. We also demonstrate that ITRQ is superior to the seemingly equivalent *Inverse iteration with Wielandt Deflation* [84] (INVWD.) The matrix used in the example corresponds to a discretized linear operator used in the stability analysis of the Brusselator wave model (BWM) [6]. Eigenvalues with the largest real parts are of interest. They help to determine the existence of stable periodic solutions to the Brusselator wave equation as a certain parameter varies. The dimension of the matrix is 200×200. The 32 rightmost eigenvalues are plotted in Figure 3.6. We place the target shift at $\sigma = 1.0$, and use ITRQ(4,5) to find 4 eigenvalues closest to σ. The TRQ equation (3.23) is solved by `GMRES(10,5)`. The residual norm of each approximate eigenpair is plotted against the number of matrix vector multiplications (MATVEC) in Figure 3.7.

It appears that the convergence of the four approximate eigenpairs occurs sequentially, i.e., the residual of the $j+1$st Ritz pair does not show significant decrease until the j-th eigenpair has been found. Since we have shown in Section 3.4.2 that an approximate inverse iteration occurs in the inexact TRQ iteration, it will be interesting

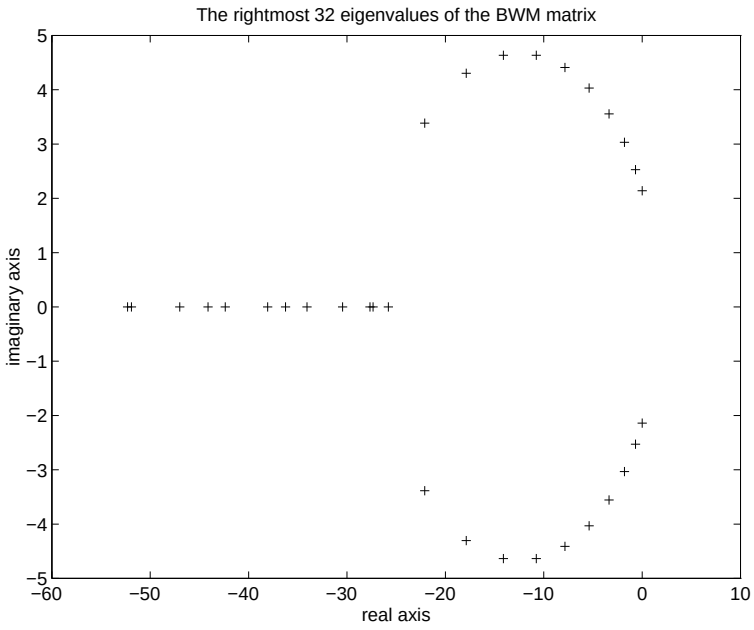

Figure 3.6 The 32 rightmost eigenvalues of a 200×200 BWM matrix.

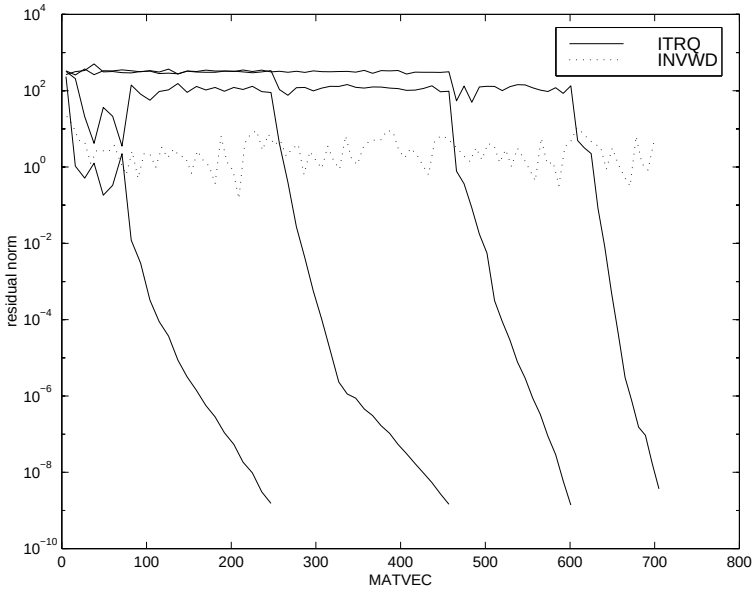

Figure 3.7 The convergence history of ITRQ.

to compare the performance of ITRQ with an accelerated inverse iteration combined with Schur-Wielandt deflation [75, pp. 117]. The INVWD algorithm computes one eigenpair at a time by an approximate inverse iteration in which the linear system $(A - \mu I)w = v$ is solved by an iterative method. Instead of continuing the inverse iteration with

$$v \leftarrow \frac{w}{\|w\|} \quad \text{and} \quad \mu = \frac{v^H A v}{v^H v},$$

we compute an Arnoldi factorization using w as the starting vector, and choose (μ, v) from the Ritz pairs associated with this factorization. This approach can also be viewed as a sequence of restarted Arnoldi iterations in which the starting vector is repeatedly enhanced by an approximate inverse iteration. Once some eigenpairs have converged, we may apply Schur-Wielandt deflation [75, pp. 117] to expose the subsequent eigenpairs. The detail of the algorithm is presented in Figure 3.8. Unlike ITRQ, INVWD must solve $(A-\mu I)w = v$. Moreover, since there is no error damping, the equation must be solved rather accurately to guarantee the convergence of the inverse iteration. In this example, we use `GMRES(20,5)`. To make a fair comparison, we use a 5-step Arnoldi factorization to accelerate the inverse iteration. We observe from Figure 3.7 that, the residual curve corresponding to INVWD (dotted curve) remains zigzag around 1.0 and never shows significant decrease.

Example 3 - The Effect of Preconditioning

The previous two examples were presented merely to illustrate that it is possible to combine an iterative solver with the TRQ iteration. We should point out that in practice both problems can be solved with an exact TRQ or a shifted and inverted Arnoldi iteration (to be discussed in the next section,) because the matrices involved in both examples can be efficiently factored.

(INVWD) A Schur-Wielandt Deflated Inverse Iteration

Input: (A, μ, v, U) such that (μ, v) is the current approximation to the desired eigenpair, and columns of U contain the converged Schur vectors.

Output: A new approximate eigenpair (μ_+, v_+) that may be used in the next cycle of an inverse iteration.

1. Solve $(A - \mu I)w = v$;
2. $v \leftarrow (I - UU^H)w$; $v \leftarrow v/\|v\|$;
3. $f \leftarrow Av$; $\alpha = v^H w$;
4. $H_1 = (\alpha)$; $V = (v)$; $f \leftarrow f - v\alpha$;
5. $f \leftarrow (I - UU^H)f$;
6. **for** $j = 1, 2, ..., k$
  6.1. $\beta_j = \|f\|$; $v_{j+1} \leftarrow f/\beta_{j+1}$;
  6.2. $V_{j+1} = (V_j,\ v_{j+1})$; $H_j \leftarrow \begin{pmatrix} H_j \\ \beta_j e_j^T \end{pmatrix}$;
  6.3. $z \leftarrow Av_{j+1}$; $z \leftarrow (I - UU^H)z$;
  6.4. $h \leftarrow V_j^H z$; $H_{j+1} = (H_j,\ h)$;
  6.5. $f \leftarrow z - V_{j+1}h$;
7. **end**;
8. Compute an desired Ritz pair (μ_+, v_+) from H_k and V_k to be used in the next cycle of an inverse iteration.

Figure 3.8 Schur-Wielandt Deflated Inverse Iteration.

As we mentioned before, the inexact TRQ is ideal for problems in which matrix factorization is prohibitively expensive. The following example carries this characteristic. We will show that with the help of a good preconditioner the speed of convergence of ITRQ can be drastically improved. The matrix used here arises from a reactive scattering calculation [59]. Its sparsity pattern is shown in Figure 3.9. Although the matrix itself has only 6% non-zero elements, a sparse matrix factoriza-

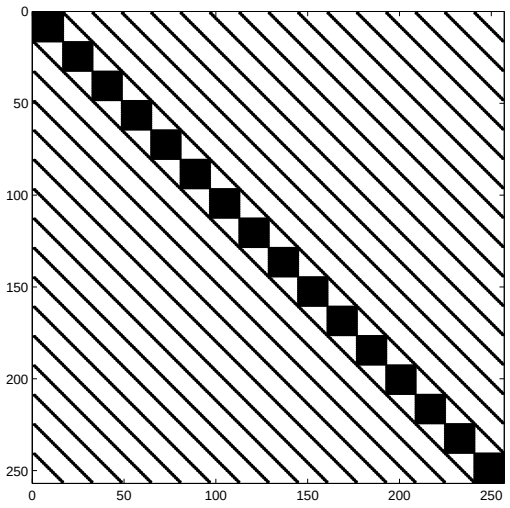

Figure 3.9 The sparsity pattern of the reactive scattering matrix.

tion tends to fill almost the entire matrix with non-zeros, regardless of the ordering scheme used.

In this application, eigenvalues near zero are of interest. For clarity, we only show the convergence of the first eigenvalue. The convergence pattern for other eigenvalues is similar to this one. The solid curve in Figure 3.10 corresponds to the residual norm of the Ritz pair obtained from running a preconditioned ITRQ(4,5). The equation (3.21) is solved by MINRES. The preconditioner used here is a matrix consisting of the dense diagonal blocks of the original matrix. Without a preconditioner, ITRQ(4,5)

performs equally well at the beginning of the iteration. As the Ritz value approaches the desired eigenvalue, the linear equation (3.21) becomes extremely difficult to solve. This explains the zig-zag behavior of its residual curve (dashed curve). We also plotted, in Figure 3.10, the residual of the Ritz approximation obtained from IRA(1,29) (dotted line.) We observe that IRA converges at a much slower rate in this case.

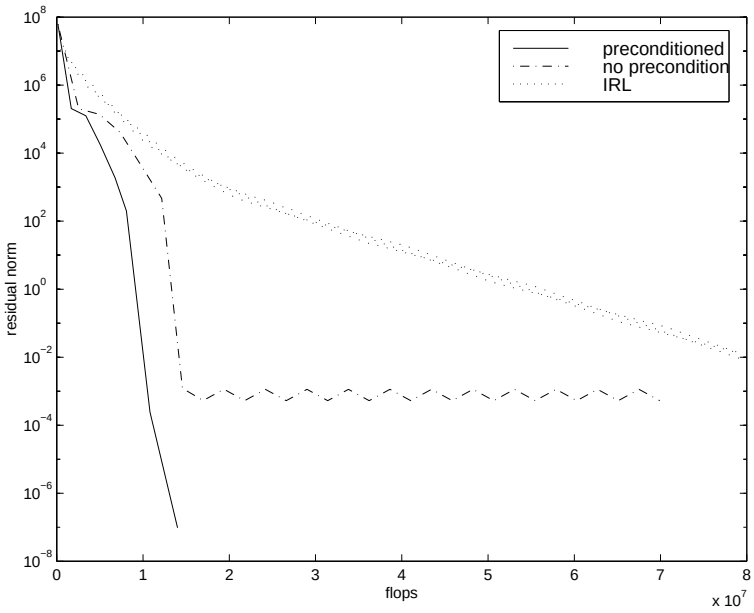

Figure 3.10 Comparison of (preconditioned) ITRQ with IRA.

Chapter 4

Spectral Transformation

In this chapter, we consider using spectral transformation to improve the convergence of the Arnoldi method. Recall that spectral transformation refers to the technique of applying the Arnoldi method to $\psi(A)$, where $\psi(\lambda)$ is some simple function. The purpose of this transformation is to convert the interior and clustered eigenvalues of A to the dominant and well separated eigenvalues of $\psi(A)$. The spectral transformation considered in this chapter is the classical one:

$$\psi(\lambda) = \frac{1}{\lambda - \sigma},$$

where σ is a *target shift*. The corresponding Arnoldi(Lanczos) iteration is often referred to as a *shifted and inverted Arnoldi(Lanczos)*. Other possibilities are the polynomial transformations to be discussed in the next chapter and the *Cayley* transform proposed in [46].

Spectral transformation is also a powerful tool for solving the generalized eigenvalue problem

$$Kx = \lambda Mx,$$

in which K or M may be singular. The problem is often transformed into

$$(K - \sigma M)^{-1} Mx = \frac{1}{\lambda - \sigma} x$$

before the Arnoldi or Lanczos algorithm is applied. In the first part of this chapter, we will discuss the practical aspect of the shifted and inverted Lanczos iteration. In particular, we will focus on problems in which a large number of eigenvalues are of interest. This is often solved by concatenating several Lanczos runs, each responsible

for acquiring a subset of the desired spectrum, in an efficient manner. (See the flow chart in Figure 4.) We provide some heuristics for minimizing the total cost associated with this technique.

A variation of the shifted and inverted Lanczos iteration is the Rational Krylov Subspace (RKS) method proposed by Ruhe [66]. The method is also suitable for applications in which eigenvalues within a wide range must be computed. Properties of this method will be studied in Section 4.2.

4.1 A Practical Use of the Shifted and Inverted Lanczos

For convenience, we will consider only the symmetric generalized eigenvalue problem in this section. When a few eigenvalues of the matrix pencil (K, M) are of interest, the shifted and inverted Lanczos is often the most effective way of solving the problem assuming one can factor the matrix $K - \sigma M$ efficiently. It is reported in [84] that a shifted and inverted IRL outperforms the TRQ iteration with a fixed shift. However, for many applications such as structure analysis [27], it is often desirable to obtain a large number of eigenvalues within a certain frequency interval, say $[\omega_1, \omega_2]$. The frequencies of interest can vary, for example, from 10^{-2} to 10^4. A shifted and inverted Lanczos run with a single-shift often does not provide the efficiency and robustness essential to a large-scale simulation. This is due to the fact that eigenvalues far away from the target shift tend to converge slowly. (See Figure 4.2.) To capture all desired eigenvalues within a single Lanczos run, one must project the shifted and inverted operator $(K - \sigma M)^{-1}M$ into a Krylov subspace of large dimension. As a consequence, the cost for maintaining orthogonality and the Ritz calculation increases significantly. Performance improvements can be made if we replace a single Lanczos run with multiple runs. Each run uses its own shift to find a subset of the desired spectrum. The termination of one particular run is issued when all eigenvalues within

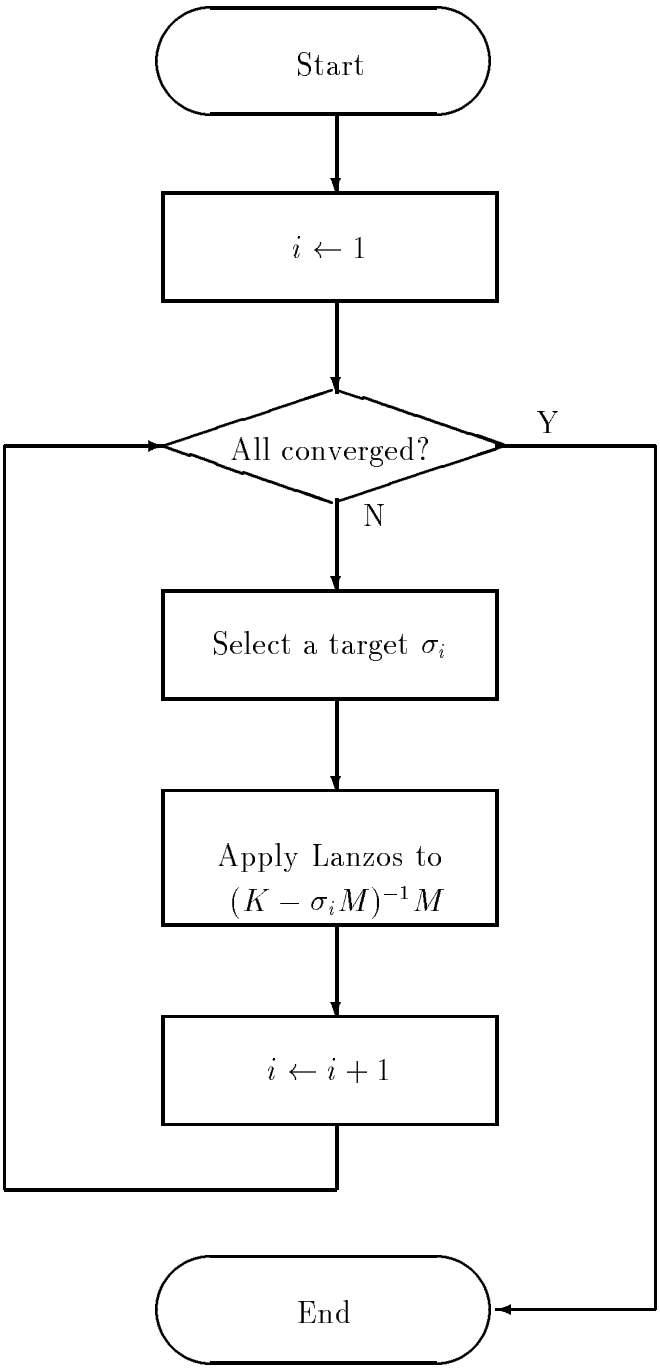

Figure 4.1 A flow chart for concatenating several Lanczos runs together.

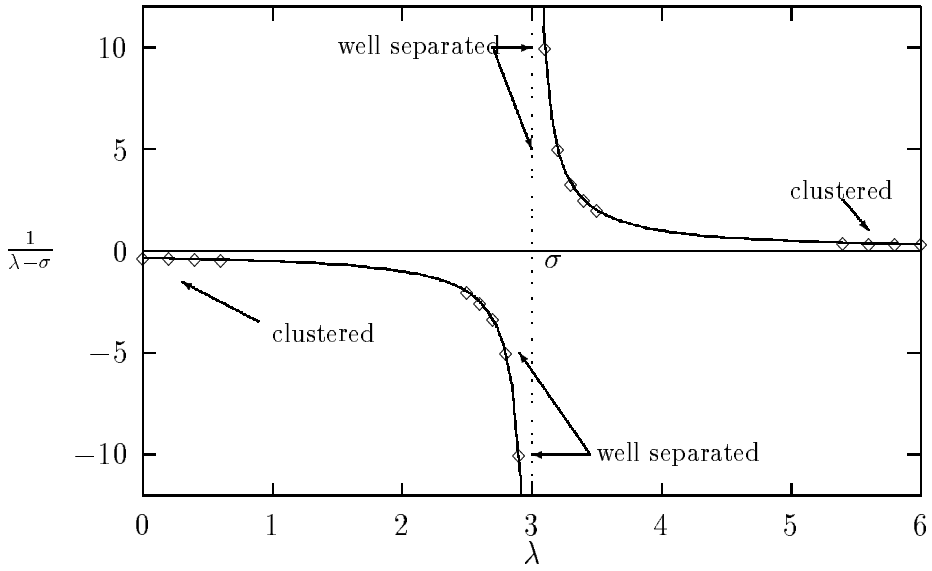

Figure 4.2 Effect of spectral transformation $\psi(\lambda) = \frac{1}{\lambda-\sigma}$.

a pre-specified subset of the spectrum have converged, or when the cost of continuing exceeds that of starting a new run with a better shift. (See Figure 4.3.)

Some care must be exercised to ensure that the union of computed subsets coincides with the portion of the spectrum of interest. With multiple matrix factorizations, this can be achieved using a few inertia counts [27]. We will say more on this topic in connection with the choice of the target shifts in Section 4.1.3.

Another important question to keep in mind is the orthogonality between the eigenvectors computed from different runs. When the eigenvalues are well-separated or when the eigenvectors are computed to high accuracy, this is usually not a big concern. This is because eigenvalues nearest to the target usually converge faster than those that are far away. However, if the eigenvalues to be computed are clustered, one should pay special attention to maintaining orthogonality between eigenvectors computed from different Lanczos runs to avoid missing multiple eigenvalues.

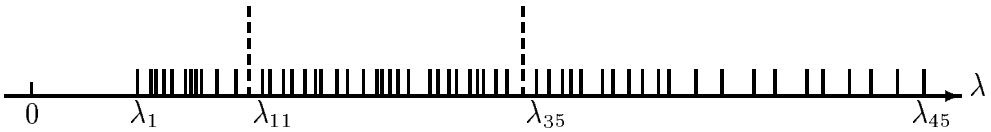

Figure 4.3 Split the spectrum of interest into several subsets and combine several Lanczos runs with different target shifts to capture all desired eigenvalues.

Although the advantage of introducing multiple target shifts is apparent from the standpoint of improving the convergence rate and reducing the cost of maintaining orthogonality, these shifts carry an additional factorization cost that must be accounted for in the evaluation of the overall computational performance. One must be very careful in deciding when to replace the current run with a new run. A good criterion should have the effect of globally minimizing the average cost for computing one eigenpair.

4.1.1 The Boeing Heuristics

In [27], heuristics are developed to help determine when to terminate the current shifted and inverted Lanczos run, and to begin a new run. The main goal is to minimize the average cost for computing one eigenpair. Towards this end, one must:

1. carefully evaluate the total cost associated with a k-step Lanczos process; and

2. accurately predict the number of eigenvalues that would emerge if the Lanczos process were continued.

The total cost of the Boeing Lanczos code depends on several factors shown in Table 4.1. (These factors are evaluated solely in term of the number of Lanczos steps.) Since each factor is a function of k, the number of Lanczos steps executed, one may

operation	cost
Ritz calculation	$\mathcal{O}(k^3)$
Gram-Schmidt orthogonalization	$\mathcal{O}(k^2)$
selective reorthogonalization	$\mathcal{O}(k^2)$
back solve	$\mathcal{O}(k)$
matrix vector multiplications	$\mathcal{O}(k)$
matrix factorization	$\mathcal{O}(1)$

Table 4.1 Components of the cost function associated with a k-step Lanczos run.

express the total cost as

$$C(k) = \alpha_0 k^3 + \alpha_1 k^2 + \alpha_2 k + \alpha_3,$$

and find the coefficients $\{\alpha_i\}_{i=0}^3$ by constructing a least square fit to the CPU timing collected at previous Lanczos steps. The constructed $C(k)$ is evaluated at the next few k values to predict the future cost of the Lanczos run.

In the meantime, one must also keep track of the number of converged eigenvalues and predict the ones that are likely to converge if the current Lanczos factorization is extended. The Boeing code assumes that Ritz values converge in a linear fashion at a rate γ. This convergence rate is estimated by taking a weighted geometric average of the change in accuracy of the first unconverged Ritz value over the previous five steps [27].

Intuitively, the average cost function

$$\hat{C}(k) = \frac{C(k)}{\text{number of eigenvalues predicted to converge}}$$

should have a minimum. At the beginning of the Lanczos run, factoring $K - \sigma M$ constitutes a major component of $C(k)$. However, since this is a one-time cost, it can be amortized over several rapidly converging eigenvalues near the target shift.

Therefore, the average cost tends to decrease at the beginning. After the 'nearby' eigenvalues have converged, the convergence rate of the subsequent eigenvalues starts to deteriorate, and the effort required to maintain orthogonality begins to dominate the computation, causing $\hat{C}(k)$ to increase. The Boeing code stops a Lanczos run when the minimum of $\hat{C}(k)$ is detected.

4.1.2 Adding Heuristics to IRL

Similar ideas can be adopted in IRL to make it suitable for computing a large number of eigenvalues within a specified interval. Instead of detecting the local minimum of the cost curve within a single Lanczos run, we developed a slightly different strategy to minimize the overall computational cost.

Recall that the IRL algorithm presented in Chapter 2 starts with a $(k+p)$-step Lanczos factorization:

$$AV_{k+p} = V_{k+p}H_{k+p} + f_{k+p}e_{k+p}^T.$$

If the number of converged Ritz values at this point is less than k, we apply p implicitly shifted QR updates using p "unwanted" Ritz values as shifts. This step is often expressed by

$$A\Big(V_{k+p}Q_{k+p}\Big) = \Big(V_{k+p}Q_{k+p}\Big)\Big(Q_{k+p}^H H_{k+p}Q_{k+p}\Big) + f_{k+p}e_{k+p}^T Q_{k+p},$$

where Q_{k+p} is a product of orthogonal transformations used in the implicitly shifted QR bulge chase. Note that the term "shift" used here is not to be confused with the target shift σ used in the spectral transformation. Upon the completion of the implicit QR updates, we have a new Lanczos factorization of length k. To complete an IRL cycle, we extend the new factorization to a length $k+p$ factorization by performing p additional Gram-Schmidt operations to obtain

$$AV_{k+p}^+ = V_{k+p}^+ H_{k+p}^+ + f_{k+p}^+ e_{k+p}^T.$$

To make IRL cost effective for the task of computing a large number of eigenvalues within some specified interval, we must modify the basic iteration slightly. In the following, we assume that eigenvalues within the interval $[\omega_1, \omega_2]$ are computed from left to right. After the initial target shift (usually $\sigma_1 \equiv \omega_1$) has been selected, we construct a length $k+p$ Lanczos factorization. If the number of converged eigenvalues is less than k , an implicit restart is always carried out. The computation performed above will be referred to as the *initial calculation* in the following discussion. No cost analysis is performed during the initial calculation. Our experiments show that the initial calculation often captures quite a few eigenvalues. This is mainly because the distance between σ_1 and the first few eigenvalues within the $[\omega_1, \omega_2]$ is typically very small, hence the spectral transformation is very effective.

We should point out that in many applications, the maximum size of the Lanczos factorization ($m = k + p$) is typically set based on the amount of memory available on the user's computer system. We initially set the number of desired eigenvalues in each run (k) to $m/2$. This value is adjusted as IRL proceeds. When convergence takes place rapidly, we increase the value of k.

Unlike the standard IRL iteration which can only be terminated after a complete restarting cycle, the modified iteration can be stopped at any point after the initial calculation. In particular, we shall exit the current IRL when the cost analysis shows that

- the convergence rate becomes slow, and the average cost for extending the current Lanczos factorization exceeds that for starting a new IRL run, or
- the polynomial filtering corresponding to the implicit QR update is not making enough progress, and average cost associated with implicit restart becomes so

high that it is better off to switch to another target shift and to begin a new IRL run.

We introduce the following notation for cost analysis:

T_e Elapsed time since the beginning of the computation.

T_n CPU time required to carry out the next action.

n_c The number of converged eigenvalues.

n_p The number of eigenvalues predicted to converge.

The main task in the analysis is to estimate the average computational cost associated with various actions to be taken. The estimation relies on the simple formula:

$$C = \frac{T_e + T_n}{n_c + n_p}. \tag{4.1}$$

The decision of whether to continue the current IRL or to start a new run is made in favor of the action that will yield the least C value.

In our code, each IRL run will carry out at least ℓ Lanczos iterations. In addition to providing the approximations to the desired eigenvalues, the initial calculation also estimates the cost of various computational components within IRL. In particular, we collect the CPU timing at the j-th step of the Lanczos process for $j = \ell, \ell+1, ..., k+p$, and find a least square fit to the measured data using a quadratic model:

$$L_j = \alpha_0 j^2 + \alpha_1 j + \alpha_2. \tag{4.2}$$

In our numerical experiments we choose $\ell = 10$. This approach is different from the cubic model used in the Boeing heuristics. The cubic term which corresponds to the Ritz vector calculation is not included here because in IRL the Ritz vectors are computed after the IRL is terminated.

Once the coefficients α_i's in (4.2) are determined, we estimate the CPU time required to extend a length j Lanczos factorization to a length $j+1$ factorization, for $j = \ell, \ell+1, ..., k+p-1$, i.e., we compute

$$\Delta_{j+1} = L_{j+1} - L_j.$$

These values will be used in place of T_n to determine the average cost of extending the Lanczos factorization. For a similar reason, the CPU time required to carry out the implicit restart is also measured during the initial calculation.

For cost analysis purposes, our heuristic assumes subsequent IRL runs will always start with a $k+p$ Lanczos factorization. This is not necessarily true in the actual computation. (One may terminate the subsequent IRL before a length $k+p$ Lanczos factorization is constructed.) Our assumption is made to set a value of T_n associated with a new IRL run. Based on this assumption T_n is taken to be L_{k+p}.

Throughout the computation we will always keep track of the number of converged eigenvalues n_c and the total elapsed time T_e. Therefore, the only missing piece in (4.1) is n_p, the number of eigenvalues predicted to converge. The estimation of n_p is the most difficult and critical part of the cost analysis.

To predict the number of eigenvalues that will emerge in a new IRL run, we need to compute a *convergence radius* during the initial calculation. The convergence radius R is defined as:

$$R = \lambda_m - \sigma_1,$$

where σ_1 is the initial target shift and λ_m is the converged eigenvalue furthest away from σ_1. Suppose we have selected a candidate for the next target shift $\hat{\sigma}$ using the strategy to be discussed in Section 4.1.3, a reasonable estimate of n_p can be obtained by counting the number of unconverged Ritz values (θ's) that satisfy

$$\left| \frac{1}{\theta} + \sigma - \hat{\sigma} \right| < R.$$

The convergence radius is updated in subsequent IRL runs if the distance between a target shift and the largest eigenvalue obtained by the first $k + p$ steps of the corresponding IRL exceeds the previous R.

Our heuristics for predicting the number of eigenvalues to converge during the next step of the current IRL is based solely on the Ritz estimates. Recall that for a j-step Lanczos process, a Ritz estimate associated with a Ritz pair (θ, z) is defined as

$$\varepsilon = \|f_j\||e_j^T y|,$$

where f_j is the residual vector associated with the Lanczos factorization currently available, and y is an eigenvector of H_j corresponding to the Ritz value θ. We predict θ to converge in the next step of IRL if

$$\varepsilon < \delta\tau^\gamma \max\{|\theta|, (\text{machine } epsilon)^{\frac{2}{3}}\},$$

where τ is the convergence tolerance set by the user,

$$\gamma = \begin{cases} \frac{1}{2} & \text{if the next step is an implicit restart} \\ \frac{3}{4} & \text{if the next step is an extension of the Lanczos factorization} \end{cases}$$

and δ is a penalty factor initialized to 1. Since the choice of γ is purely heuristic, the above prediction could be incorrect. The penalty factor δ is used to correct an erroneous prediction. It is increased when under-prediction occurs and decreased when over-prediction is detected.

4.1.3 How to Choose the Target Shift

The choice of target shifts plays an important role in combining several Lanczos runs together. We want to choose shifts that will yield effective spectral transformations. The effectiveness of a spectral transformation is measured by the number of eigenvalues that emerge in the corresponding shifted and inverted Lanczos.

In our modified IRL, eigenvalues are computed from left to right, that is to say, in each IRL we filter out eigenvalues to the left of the target shift σ and compute only the ones that are greater than σ. This approach makes our target shift selection strategy much simpler than the ones presented in [27]. The basic idea is to place the target shift between the rightmost converged eigenvalue λ_r and the leftmost Ritz value θ that has not converged. Figure 4.4 illustrates the position of σ.

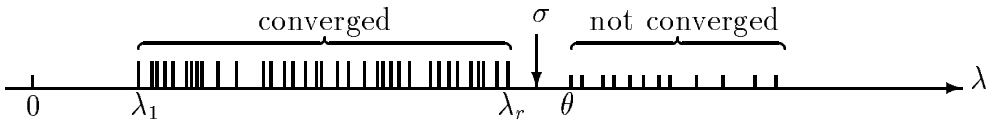

Figure 4.4 The location of the next target shift.

The disadvantage of this approach is that we are only keeping approximately half of the eigenvalues that would emerge rapidly from a shifted and inverted Lanczos. A more efficient strategy is to place the shift further to the right so that eigenvalues from both sides of σ can be captured at the same time. (See Figure 4.5.) However, to

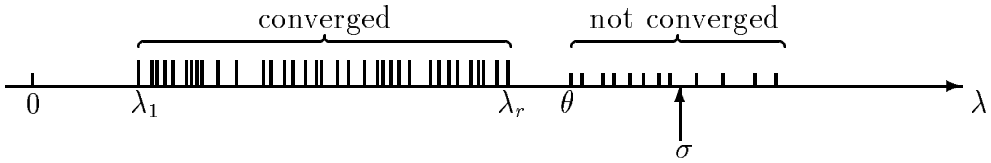

Figure 4.5 Another way of choosing the next target shift.

avoid recomputing the eigenvalues that have converged, one must use either deflation or implicit shifting to remove the contribution of the converged eigenvalue from the next IRL run. A more severe difficulty arises when the target shift is placed too far to the right. If the next IRL is not able to capture all remaining eigenvalues between the current and the previous shifts, an extra IRL run must be used to find the missing

ones. This is the main reason that we choose to use the conservative, but simpler scheme of putting σ adjacent to λ_r.

If there is large gap between λ_ℓ and θ, we try to place the target shift closer to θ to make the spectral transformation more effective. To be more specific, we determine σ from the formula

$$\sigma = \frac{w_1\lambda_r + w_2\theta}{w_1 + w_2},$$

where w_1 and w_2 are some weights. In our experiments, we choose $w_1 = 1$ and $w_2 = 5$. This is merely a heuristic. Other combinations of w_1 and w_2 are possible.

It is possible that the unconverged Ritz values interlace with the converged ones as shown in Figure 4.6. In this case, we set $\sigma = \theta$, the leftmost unconverged Ritz

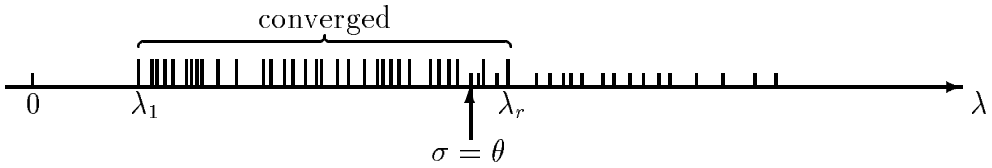

Figure 4.6 Ritz values interlace with converged eigenvalues.

value. Since we do not want to recompute the converged eigenvalues larger than σ, deflation must be used in the next IRL run. That is, we apply IRL to the projected matrix

$$(I - ZZ^H)(K - \sigma M)^{-1}M(I - ZZ^H),$$

where Z is a matrix consisting of eigenvectors corresponding to the the converged eigenvalues to the right of σ. These eigenvectors normally reside on the secondary storage device (disk), and must be read into the fast memory before deflation can take place.

If the gap between λ_r and θ is extremely small, the last few converged eigenvalues are likely to be in a tight cluster. To avoid missing eigenvalues within the cluster,

we place the shift between the last two converged eigenvalues. This is illustrated in Figure 4.7. Again, deflation is necessary to remove λ_r from the next IRL run.

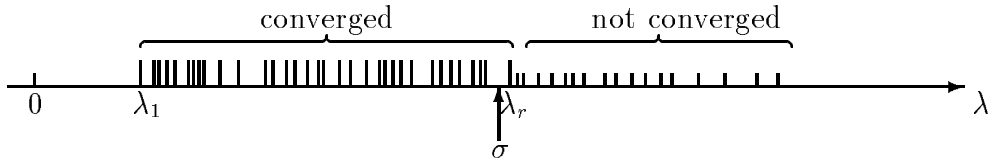

Figure 4.7 Place the target shift between the last two converged eigenvalues to avoid missing clustered eigenvalues.

If the cost analysis indicates that it is time to start a new IRL run, we choose a target shift using the above strategy. We can determine from the Cholesky factorization

$$K - \sigma M = LDL^T,$$

the number of eigenvalues to the left of σ by simply counting the negative entries in the diagonal matrix D. This is often referred to as an *inertia count*. The inertia count can be used to verify that there is no eigenvalue missing during the previous computation. If the number of converged eigenvalue is less than the inertia count, it is clear that some eigenvalues to the left of σ have not been captured. Again, this typically occurs where some eigenvalues are tightly clustered. These eigenvalues tend to converge at a slower rate than other eigenvalues including the ones to the right. When this situation occurs we move σ back (towards the left.) This will cost an additional matrix factorization. An alternative is to change the shifting strategy in IRL to allow eigenvalues from both sides σ to be computed. However, this approach requires one to deflate all eigenvectors that have been computed, which is also costly.

4.1.4 Numerical Experiment

In the following, we present an example of combining several IRL runs to extract the lowest 100 eigenvalues of a vibration model for an ore car. The matrix pair BCSSTK11 and BCSSTM11 used are taken from the Harwell-Boeing matrix collection [17]. The order of these matrices is $n = 1473$. Eigenvalues of interest lie between $\omega_1 = 10.5$ and $\omega_2 = 1.7142 \times 10^4$. The computation is performed on a Silicon Graphics Power Challenge (R8000) machine in double precision. A tight convergence tolerance $tol = 1.0^{-15}$ was used.

We will refer to an IRL run with a fixed target shift and a large $k + p$ value as a single-shift IRL run. In this problem, a single-shift Lanczos run must project $(K - \sigma M)^{-1}M$ into a Krylov subspace of dimension at least 100 in order to capture all desired eigenvalues. In `ARPACK`, this amounts to allocating at least $1473 \times 101 \times 8 = 1.2$ mega bytes for the Lanczos vectors in addition to the storage required for matrix factorization and other internal work space (25k bytes) needed to carry out the Lanczos iteration. With multiple target shifts, not only can the storage be saved, the amount of computational time can be reduced as well.

Table 4.2 compares the performance of a single-shift IRL run with that of concatenating several IRL runs with different target shifts. In the single-shift run, we set $k + p = 120$ and $p = 20$. In the test that allows multiple target shifts, we use $k + p = 60$ and $k = 40$. We observe that the multiple-shift run outperformed the single-shift run in CPU time by 15%. The multiple-shift run is also designed to take full advantage large memory space if available. Thus, a large $k + p$ value will usually result in less CPU time as indicated in Figure 4.8.

Table 4.3 gives a detailed account of the shifting process that went on during the entire computation. The leftmost column shows the target shifts used in the compu-

	NCV	CPU time (seconds)
single-shift	120	15.2
multiple-shift	60	13.0

Table 4.2 Comparison of the single-shift ARPACK run with the multiple-shift variant.

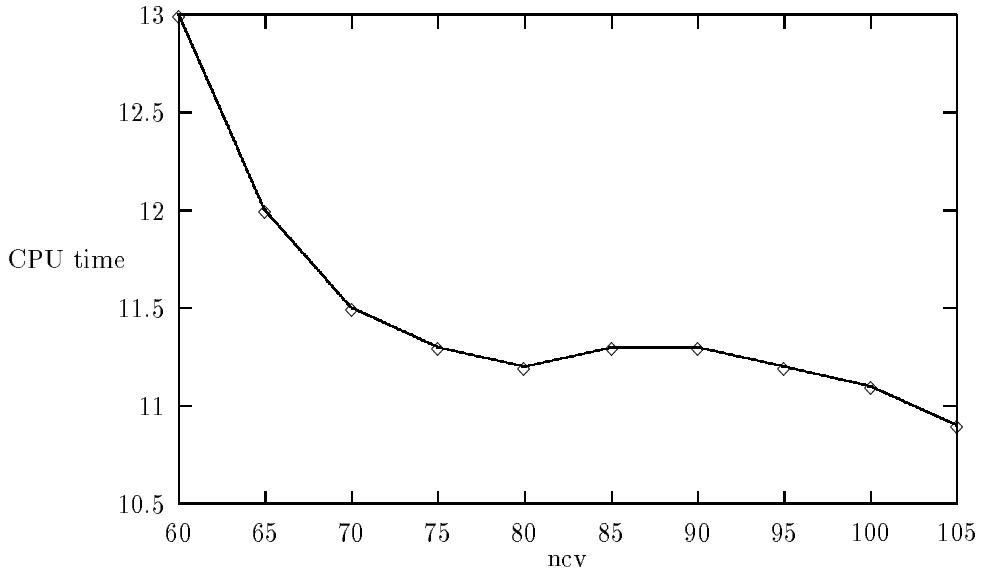

Figure 4.8 Increasing the dimension of the Krylov subspace $(k + p)$ lowers the overall computational cost of the multi-shift Lanczos driver.

tation. Each shift is associated with a shifted and inverted IRL run consisting of a number of implicit restarts. To make this example easier to understand, we did not allow IRL to terminate before a restarting cycle was completed. The number of Ritz values predicted to converge (column 3) is compared to the number of eigenvalues actually converged (column 4) in each restart to illustrate the accuracy of our prediction scheme. Column 5 lists the estimated average cost associated with either a restart or a new IRL run. The actual cost are shown in Column 6 It is easy to see that the discrepency between the last two columns is small indicating that our cost analysis model is quite effective. Since we have limited the dimension of the Krylov subspace

shift	restart number	$n_{predict}$	$n_{converged}$	predicted cost	actual cost
0.0	1	–	24	–	0.056
	2	6	6	0.062	0.051
	3	2	2	0.060	0.063
	4	9	7	0.061	0.061
	5	7	1	0.061	0.068
5.1×10^3	1	6	8	0.085	0.087
	2	8	1	0.087	0.092
	3	3	1	0.098	0.097
	4	2	2	0.104	0.100
8.9×10^3	1	5	9	0.114	0.106
	2	9	0	0.103	0.113
	3	5	2	0.112	0.114
1.2×10^4	1	8	8	0.117	0.119
	2	8	0	0.117	0.125
1.5×10^4	1	11	6	0.123	0.133
1.6×10^4	1	16	10	0.126	0.133
1.7×10^4	1	13	12	0.131	0.130

Table 4.3 Detailed view of the multi-shift Lanczos process

to $k + p = 60$. The cost of orthogonalization constitutes only 10% of the total cost. (See Table 4.4.) This is in sharp constrast with the Boeing sparse eigenvalue solver

in which the cost of reorthogonalization constitutes 30% of the total cost in general [27].

operation	percent
factorization	29
back solve	41
matrix vector multiplication	4
eigenvector calculation	7
Gram-Schmidt orthogonalization	8
reorthogonalization	5
QR update	6

Table 4.4 Percentage of CPU time spent in various parts of the computation.

Although the basic idea of spectral transformation is simple. Developing an optimal strategy for combining several shift-invert Lanczos runs to achieve the goal of obtaining a large number of desired eigenvalues within a specific interval is a challenging task. The design and implementation of the shifting strategy described above is still preliminary. Further experiments are necessary to refine the cost analysis model presented earlier.

An alternative to combing several shift-invert Lanczos processes together is to vary the shift within a single Lanczos run. In this case, the matrix pencil is projected into a subspace $\mathcal{S}$ that can be described as

$$\mathcal{S} = \{w_1, w_2, \ldots, w_k\}, \quad \text{where} \quad w_{j+1} = (K - \sigma_j)^{-1} M w_j.$$

Since $\mathcal{S}$ is not a Krylov subspace, the algorithm associated with this construction is not really a standard Lanczos algorithm. This algorithm is first proposed by Ruhe [66] and is named the *Rational Krylov Sequence* method. In the next section, we will present this algorithm for completeness.

4.2 The Rational Krylov Method

The need for computing several eigenvalues within a large interval (or a large region within the complex plane) gives motivation to the development of the *Rational Krylov Sequence* (RKS) method of Ruhe [66, 68, 67]. The RKS method can also be used as a tool to construct a reduced order model for a large scale single input single output (SISO) linear (control) system [25].

Given a sequence of shifts σ_i, the RKS method mimics the shift-invert Arnoldi iteration by generating an orthonormal basis of a *Rational Krylov Subspace*

$$\mathcal{S} = \{w_1, w_2, ..., w_k\}, \quad \text{where} \quad w_{j+1} = (K - \sigma_j M)^{-1} M w_j. \tag{4.3}$$

Each new basis vector is generated by applying $(K-\sigma_j M)^{-1}M$ to a linear combination of several previously generated basis vectors and orthonormalizing the result against all previous basis vectors. To avoid slow convergence caused by eigenvalues far away from the initial target, the shift σ_i used in a RKS iteration is allowed to change from step to step. To see the consequence of this change, let us examine algebraic properties these basis vectors must satisfy.

Suppose an orthonormal basis $V_j \in \mathbb{C}^{n\times j}$ for a j-dimensional rational Krylov subspace is available. To generate the next basis vector v_{j+1}, one performs

$$\begin{aligned} f_j &\leftarrow (I - V_j V_j^H)(K - \sigma_j M)^{-1} M (V_j t_j); \\ \beta_j &= \|f_j\|; \\ v_{j+1} &\leftarrow f_j/\beta_j, \end{aligned}$$

where the vector $t_j \in \mathbb{C}^{j\times 1}$ can be chosen arbitrarily. If we put

$$h_j = V_j^H (K - \sigma M)^{-1} M (V_j t_j),$$

it follows from above that

$$K V_j h_j + K v_{j+1}\beta_j = M(V_j t_j) + \sigma_j M V_j h_j + \sigma_j M v_{j+1}.$$

The cumulative result of this procedure can be described by the following matrix equation

$$KV_{j+1}\hat{H}_{j+1} = MV_{j+1}\hat{G}_{j+1}, \tag{4.4}$$

where

$$\hat{H}_{j+1}e_i = \begin{pmatrix} h_i \\ \beta_i \\ 0 \end{pmatrix} \quad \text{and} \quad \hat{G}_{j+1}e_i = \begin{pmatrix} \sigma_i h_i + t_i \\ \beta_i \sigma_i \\ 0 \end{pmatrix}.$$

The details of the algorithm are given in Figure 4.9.

(RKS) Rational Krylov Sequence Iteration

Input: (A, v_1) such that $\|v_1\| = 1$.
Output: $(V_{k+1}, \hat{H}_k, \hat{G}_k)$ such that $AV_{k+1}\hat{H}_k = V_{k+1}\hat{G}_k, V_{k+1}^H V_{k+1} = I$.

1. Choose $t_1 = e_1$;
2. $V_1 \leftarrow (v_1)$; $\hat{H}_0 = (\,)$; $\hat{G}_0 = (\,)$;
3. **for** $j = 1, 2, 3, ..., k$.
  3.1. Choose a shift μ_j;
  3.2. $f_{j+1} \leftarrow (A - \mu_j I)^{-1}(V_j t_j)$;
  3.3. $h_j \leftarrow V_j^H f_j$; $\hat{H}_j \leftarrow (\hat{H}_{j-1},\ h_j)$;
  3.4. $g_j \leftarrow h_j \mu_j + t_j$; $\hat{G}_j \leftarrow (\hat{G}_{j-1},\ g_j)$;
  3.5. $f_{j+1} \leftarrow f_{j+1} - V_j h_j$; $\beta_j = \|f_{j+1}\|$;
  3.6. $\hat{H}_j \leftarrow \begin{pmatrix} \hat{H}_j \\ \beta_j e_j^T \end{pmatrix}$; $\hat{G}_j \leftarrow \begin{pmatrix} \hat{G}_j \\ \mu_j \beta_j e_j^T \end{pmatrix}$;
  3.7. $v_{j+1} \leftarrow f_{j+1}/\beta_j$; $V_{j+1} \leftarrow (V_j,\ v_{j+1})$;
  3.8. Choose a vector t_{j+1};
4. **end**

Figure 4.9 Rational Krylov Sequence Iteration.

There are several possible ways to extract approximate eigenvalues and eigenvectors from the rational Krylov subspace. The original scheme proposed by Ruhe is to

compute the generalized eigenpairs of the pencil (H_j, G_j), where H_j and G_j consist of the first j rows of $\hat{H}_{j+1}$ and $\hat{G}_{j+1}$ respectively. An alternative method suggested in [39] is to put an approximate eigenvector in the form $z = V_{j+1}\hat{H}_j s_j$ and to invoke the Galerkin criterion:

$$(V_{j+1}\hat{H}_j)^H \Big(M^{-1}Kz - \theta z \Big) = 0.$$

Using (4.4), the left hand side of the above equation can be further simplified by

$$\begin{aligned} (V_{j+1}\hat{H}_j)^H \Big(M^{-1}KV_{j+1}\hat{H}_j s_j - \theta V_{j+1}\hat{H}_j s_j \Big) &= \hat{H}_j^H V_{j+1}^H \Big(V_{j+1}\hat{G}_j s_j - \theta V_{j+1}\hat{H}_j s_j \Big) \\ &= \hat{H}_j^H \hat{G}_j s_j - \theta \hat{H}_j^H \hat{H}_j s_j \end{aligned}$$

If

$$\hat{H}_j^H \hat{H}_j s_j = \theta \hat{H}_j \hat{G}_j s_j,$$

θ is said to be a Ritz value and $z = V_{j+1}\hat{H}_j s_j$ is said to be a Ritz vector associated with the rational Krylov subspace generated. Note that the rational Krylov subspace $\mathcal{S}$ is completely different from the standard Krylov subspace. Its connection with the Jacobi-Davidson method (to be described in Chapter 6) has been exploited in [69, 39].

Many practical issues are yet to be resolved to make the RKS method reliable and efficient. For example, it is suggested in [69] that one should use the same shift for several steps to reduce the factorization cost. However, no heuristic is available to help decide when to use a new shift. The cost of orthogonalization becomes a significant part of the computational expense when the rational Krylov subspace becomes large. Some purging and restart mechanisms have been proposed in [69]. The main idea is to remove the converged Schur vectors from the rational Krylov subspace and work with only those vectors that contribute to the calculation of unconverged eigenvalues.

Chapter 5

Polynomial Spectral Transformation

Although applying the Arnoldi/Lanczos method to $(A-\mu I)^{-1}$ can be extremely helpful for finding the interior eigenvalues of A, one must factor the matrix $A-\mu I$ and solve linear systems directly. When the direct methods for solving the linear system become prohibitively expensive, one may consider combining an iterative solver with the TRQ or the Jacobi-Davidson method (to be discussed in the next chapter) to tackle the interior eigenvalue problem. In general, both the inexact TRQ and Jacobi-Davidson method require good preconditioners to be effective. When these preconditioners are not available, convergence is often difficult to predict.

Another factorization free alternative is to replace $(A-\mu I)^{-1}$ with $p(A)$ where $p(\lambda)$ is a low degree polynomial that retains the spectral enhancing property of the rational function $\frac{1}{\lambda-\mu}$.

The idea of using polynomial transformations to accelerate the Lanczos type of algorithm is not new. In fact, Stiefel proposed using Kernel polynomials to accelerate the calculation of interior eigenvalues in 1953 [85]. Since then, a large amount of work has been done on constructing polynomial preconditioners for conjugate-gradient-like iterative solvers [32, 72, 74, 3, 4, 18]. Although these polynomials are used in a different context, the underlying design principle remains the same.

The main advantages of a polynomial transformation are its simplicity and flexibility. Only matrix-vector multiplications are required to carry out the transformation. It is especially suitable for vector and/or parallel machines if the matrix-vector multiplication can be easily vectorized or parallelized. It is also ideal for machines on which inner product calculation is a bottleneck. This is because the accelerated

Lanczos typically converges in fewer steps, and thus computes fewer inner products than the one not being accelerated.

In this Chapter, we will focus on symmetric eigenvalue problems and investigate various types of polynomial transformations. The presentation will begin with a review of some simple polynomials that can be written in closed form. A well known example is the Chebyshev polynomial. We then focus on using *minmax* and *least squares* techniques directly to construct polynomial approximation to the "ideal" transformation function $\frac{1}{\lambda}$. These polynomials are very helpful in finding the smallest eigenvalues of a positive definite matrix. For interior eigenvalue calculation, a polynomial that has a spike near some target shift μ is often desirable. These polynomials can be built by fixing the magnitude of $p(\mu)$ and minimizing $p(\lambda)$ over an interval that encloses the spectrum of A. Again, this can be done using the tools of minmax or least squares approximation.

5.1 Chebyshev and Kernel polynomials

In this section, we consider the prototype problem of computing a few of the smallest eigenvalues of a positive definite matrix A. This problem is important in a number of applications [59]. For convenience, we assume that the matrix A has been scaled so that its largest eigenvalue is $\lambda_n = 1$. Consequently, the difficulty of the eigenvalue problem is indicated by the magnitude of the smallest eigenvalues and gap between them. A well known acceleration scheme is based on a polynomial spectral transformation of the form

$$C_m(\lambda;\alpha,\beta) = T_m\Big(\frac{\alpha+\beta-2\lambda}{\alpha-\beta}\Big),$$

where $T_m(\lambda)$ is the standard Chebyshev polynomial defined as

$$T_m(\lambda) = \begin{cases} \cos(m \cos^{-1}(\lambda)), & |\lambda| \leq 1 \\ \cosh(m \cosh^{-1}(\lambda)), & |\lambda| > 1 \end{cases}$$

or recursively as

$$T_m(\lambda) = 2\lambda T_{m-1}(\lambda) - T_{m-2}(\lambda),$$

with $T_0(\lambda) = 1$ and $T_1(\lambda) = \lambda$.

Since the polynomial $C_m(\lambda; \alpha, \beta)$ is bounded within $[\alpha, \beta]$ and increases rapidly outside of this region, applying the Lanczos algorithm to $C_m(\lambda; \alpha, \beta)$ attenuates the contribution of the unwanted eigencomponents to the subspace spanned by V_k. (See Figure 5.1.) Ideally, one should choose α and β so that the interval $[\alpha, \beta]$ encloses most of unwanted eigenvalues of A. However, in practice, optimal α and β are not known a priori since they depend on the spectrum of A. These parameters must be determined adaptively as more information is gathered about the spectrum of A. The parameter β can often be computed by applying the Lanczos algorithm to A directly, which returns a good approximation to the largest eigenvalue of A in a few iterations. There are several ways to estimate α. One could certainly choose it to be θ_{k+1}, the $k+1$st Ritz value obtained in an ℓ-step ($\ell > k$) Lanczos run (without spectral transformation.) However, this simple scheme may not work well if θ_{k+1} is too close to other Ritz values, or if θ_{k+1} is a poor approximation to an eigenvalue.

An alternative way to obtain α is to set

$$C_m(\omega; \alpha, \beta) = \Delta,$$

where ω is a lower bound of the spectrum (for example, $\omega = 0$,) and Δ is a large number (for example, $\Delta = 10.0$.) The value of α can be determined by solving (5.1).

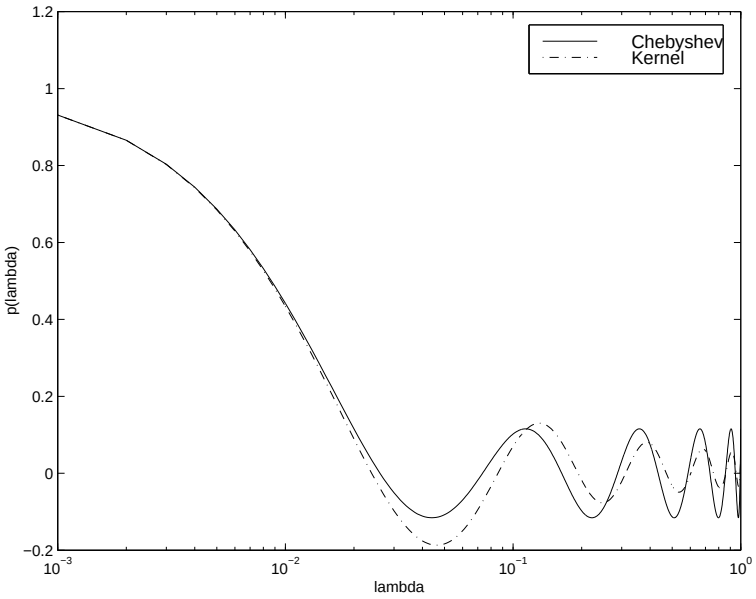

Figure 5.1 The solid curve corresponds to a 10-th degree Chebyshev polynomial with $\alpha = 0.02$ and $\beta = 1$. The dash-dot curve corresponds to a 10-th degree polynomial $K(\lambda; 0)$ constructed using Chebyshev polynomials with $\alpha = 0.01$ and $\beta = 1$.

This yields

$$\alpha = \frac{\left[1 + \cosh\left(\frac{\cosh^{-1}(\Delta)}{m}\right)\right]\beta - 2\omega}{\cosh\left(\frac{\cosh^{-1}(\Delta)}{m}\right) - 1},$$

where $\Delta \gg 1.0$. By choosing a large Δ, this approach guarantees that there is enough separation between the low and high end of the transformed spectrum. However, if Δ is too large, the constructed polynomial decreases slowly near the low end of the spectrum causing the transformed eigenvalues to be clustered also. This is illustrated in Figure 5.2.

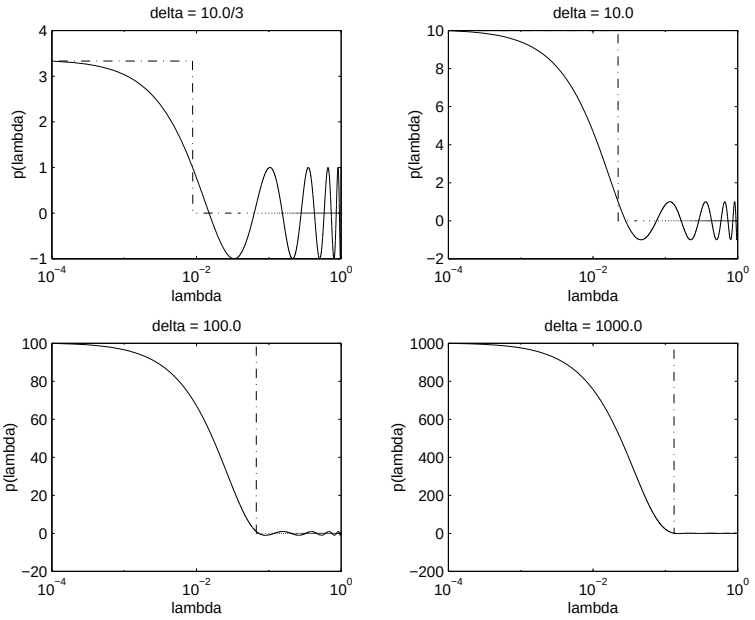

Figure 5.2 Chebyshev polynomials $C_{10}(\lambda; \alpha, \beta)$, where $\beta = 1.0$ and α is determined from (5.1) for various values of Δ. The vertical dash-dotted line indicates the location of α.

A related but different class of polynomials that has the same transformation property is the *Kernel* polynomial. Given a sequence of orthogonal polynomials $\{\phi_j\}_{j=0}^{m}$

with respect to some weighted inner product $\langle \cdot, \cdot \rangle_w$ defined on an interval $[\alpha, \beta]$:

$$\langle f, g \rangle_w = \int_\alpha^\beta f(\lambda) g(\lambda) w(\lambda) d\lambda,$$

one can express an m-th degree Kernel polynomial as

$$\hat{K}_m(\lambda; \xi) = \sum_{j=0}^{m} \phi_j(\lambda) \phi_j(\xi).$$

The kernel polynomial has many interesting properties [86]. One particularly useful property is that $\hat{K}_m(\lambda; \xi)$ has large magnitude at $\lambda = \xi$. For convenience, we normalize the polynomial by $\hat{K}(\xi; \xi)$, and define

$$K_m(\lambda; \xi) = \frac{\hat{K}_m(\lambda; \xi)}{\hat{K}_m(\xi; \xi)}$$

such that $K_m(\xi; \xi) = 1$. It can be easily shown [85] that *among all polynomials of degree less than or equal to m and having value 1 at the given point $\lambda = \xi$, the polynomial $K_m(\lambda; \xi)$ yields the least 2-norm, where the 2-norm is defined as*

$$\| K_m(\lambda; \xi) \|_2 = \Big(\int_\alpha^\beta K_m(\lambda; \xi)^2 w(\lambda) d\lambda \Big)^{\frac{1}{2}}.$$

A common choice for $\{\phi_j(\lambda)\}$ is the Chebyshev polynomial $\{C_j(\lambda; \alpha, \beta)\}$. A kernel polynomial designed in this fashion is compared with a Chebyshev polynomial in Figure 5.1.

Unlike the Chebyshev polynomial $C_m(\lambda; \alpha, \beta)$ which oscillates uniformly on $[\alpha, \beta]$, the relative extrema of the Kernel polynomial $K(\lambda; \xi)$ decreases monotonically as $\lambda \to \infty$. In addition, the magnitude of $K(\lambda; \xi)$ appears to be less sensitive to α than the Chebyshev polynomial with the same degree. In Figure 5.3, we plot several Kernel polynomials and Chebyshev polynomials with different α's. It is observed that the graph of the Kernel polynomial does not vary much as α becomes small, whereas the Chebyshev polynomial sacrifices the magnitude separation for the steepness near

zero. We number the relative extrema of each polynomial from left to right by $1, 2, \ldots$. It is observed that all but the first relative extrema of the Kernel polynomial are bounded by 0.2 (dotted horizontal line in the figure). The Chebyshev polynomials corresponding to $\alpha \leq 0.01$ exhibit larger ripple size. Therefore, when it is difficult to determine an α value, the Kernel polynomial is preferred.

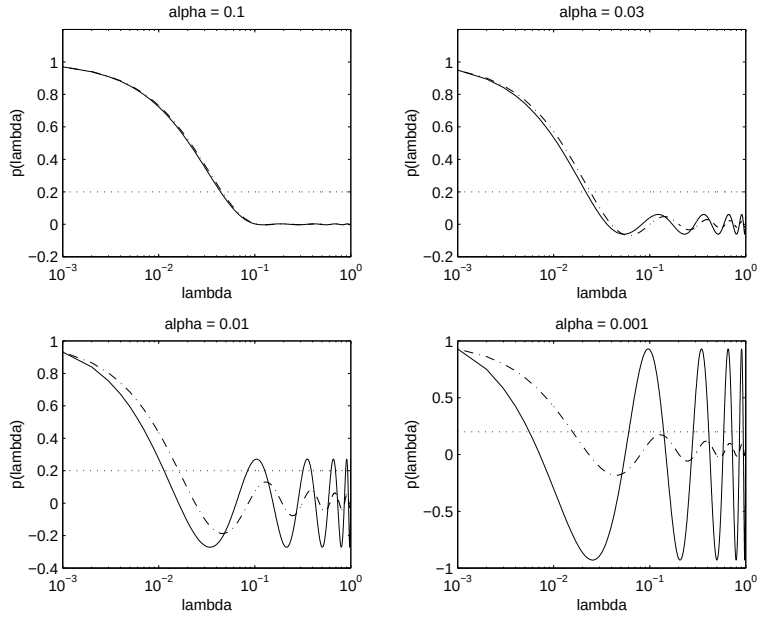

Figure 5.3 Kernel and Chebyshev polynomials on $[0.001, 1]$ with different α's. Solid curves correspond to Chebyshev polynomials and dash-dotted curves correspond to Kernel polynomials. All Chebyshev polynomials are scaled so that they have the same magnitude as the corresponding Kernel polynomial at $\lambda = 0.001$.

5.2 Leja Points and Related Polynomials

In this section, we briefly discuss polynomials associated with *Leja points*. Given a positive weight function $w(z)$ defined on a compact set $\mathbf{K}$, the Leja points associated

with $\mathbf{K}$ can be generated recursively as follows (also see Section 2.3 and [41]):

$$\begin{aligned} z_0 &\leftarrow \text{ point of largest magnitude of } \mathbf{K}; \\ z_j &: \text{ are chosen such that } w(z_j)\prod_{\ell=0}^{j-1}(z_j - z_\ell) = \max_{z\in K} w(z)\prod_{\ell=0}^{j-1}(z - z_\ell). \end{aligned}$$

Intuitively, Leja points are uniformly distributed in $\mathbf{K}$ with respect to the weight $w(z)$. Additional properties of Leja points are examined in detail in [41] [62].

One may use Leja points in many different ways to construct a polynomial transformation for accelerating the convergence of the Lanczos process. For example, if $w(z) = 1$ and $[\alpha, \beta]$ contains the entire spectrum of A, a polynomial that interpolates $\frac{1}{\lambda}$ at the Leja points of $[\alpha, \beta]$ is a good candidate. In [62], Reichel proposed using this type of polynomial to accelerate conjugate gradient type of methods for solving a linear system.

One may also use Leja points to construct transformations that have the same damping effect as that of a Chebyshev polynomial. Suppose $[\alpha', \beta]$ $(\alpha' < \beta)$ contains most of the unwanted eigenvalues of A. By placing the roots of the polynomial at the Leja points of $[\alpha', \beta]$, one obtains a polynomial almost identical to the Chebyshev polynomial $C(\lambda; \alpha', \beta)$. We will call this polynomial a *Leja damping polynomial*. Of course, one must select a suitable α' to produce an effective damping. An alternative design which does not require the estimation of an optimal α' uses Leja points associated with the weight function $w(\lambda) = |\lambda - \alpha|$. The weight function pushes Leja points towards β, yielding a polynomial that filters out the contribution of the large eigenvalues of A. We will refer to this type of polynomial as the *weighted Leja damping polynomial*. In Figure 5.4, we display polynomials constructed by three techniques described above. All polynomials have degree 10. We chose $\alpha = 0.001$, $\beta = 1$, and scaled these polynomials so that they have the same magnitude at $\lambda = \alpha$. We observe that all three polynomials share the same qualitative behavior of having a large

magnitude near $\alpha = 0.001$. The weighted Leja polynomial is steeper near $\alpha = 0.001$. Therefore, it is superior to the other two polynomials when the desired eigenvalues of A are clustered near $\alpha = 0.001$.

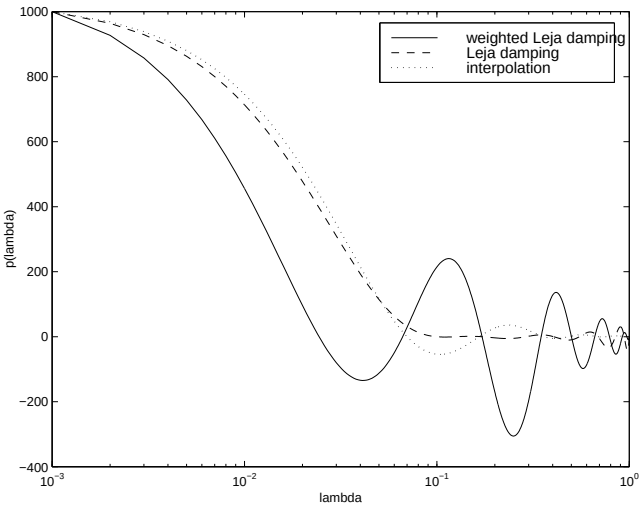

Figure 5.4 Polynomials constructed with Leja points. The solid curve corresponds to a polynomial whose roots are placed at Leja points associated with the weight function $w(\lambda) = |\lambda - 0.001|$. The dotted curve is produced by interpolating $\frac{1}{\lambda}$ at the Leja points associated with $w(\lambda) = 1$. The dashed curve represents a polynomial whose roots are the Leja points of $[0.1, 1]$ associated with the weight function $w(\lambda) = 1.0$.

5.3 Minmax Approximation and the Remez Algorithm

Before proceeding to discuss other types of polynomial accelerants, we briefly review the Remez algorithm, a numerical procedure for computing a minmax polynomial approximation. The method is widely used as a tool for filter design in digital signal processing [56]. It has also been used successfully in designing polynomial preconditioners for solving linear systems [32, 72, 74, 3, 4, 18].

A minmax (uniform) polynomial approximation to a function $f(\lambda)$ is defined as

$$\min_{\lambda\in(\alpha,\beta),p(\lambda)\in\mathcal{P}_m} \|f(\lambda)-p(\lambda)\|_\infty,$$

where $\|\cdot\|_\infty$ is defined by

$$\|f(\lambda)\|_\infty = \max_{\alpha\le\lambda\le\beta} |f(\lambda)|,$$

and $\mathcal{P}_m \equiv \{\text{polynomials of degree less than or equal to } m\}$. The solution to this approximation problem can be characterized by the following theorem often known as the *Alternation* theorem.

Theorem 5.1 *(Alternation)* Let $\{\phi_0(\lambda),\phi_1(\lambda),...,\phi_m(\lambda)\}$ be a basis of the polynomial subspace $\mathcal{P}_m$ and $f(\lambda)$ be continuous on $[\alpha,\beta]$. The minmax polynomial approximation

$$p_m(\lambda) = \sum_{j=0}^{m} \gamma_j\phi_j(\lambda), \tag{5.1}$$

to $f(\lambda)$ from $\mathcal{P}_m$ must satisfy the "alternation" criterion, that is, there must be at least $m+2$ points $\lambda_0<\lambda_1<...<\lambda_{m+1}\in[\alpha,\beta]$ such that

$$f(\lambda_k)-p_m(\lambda_k) = (-1)^k\epsilon, \quad k=0,1,...,m+1, \tag{5.2}$$

where

$$|\epsilon| = \max_{\lambda\in[\alpha,\beta]} |f(\lambda)-p_m(\lambda)|. \tag{5.3}$$

Based on this characterization, Remez [63] devised an iterative procedure that repeatedly modifies a sequence of "alternation" points $\{\lambda_j\}$ until (5.2) and (5.3) are satisfied.

The Remez algorithm begins with defining a fine mesh on $[\alpha,\beta]$ and arbitrarily selecting $n+2$ distinct *reference* points from $[\alpha,\beta]$. The algorithm enforces the

alternating condition (5.2) to hold at these points for some unknown ϵ. This amounts to solving an $(m+2) \times (m+2)$ linear system

$$\begin{pmatrix} \phi_{0,0} & \phi_{0,1} & \cdots & \phi_{0,m} & 1 \\ \phi_{1,0} & \phi_{1,1} & \cdots & \phi_{1,m} & -1 \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ \phi_{m,0} & \phi_{m,1} & \cdots & \phi_{m,m} & (-1)^m \\ \phi_{m+1,0} & \phi_{m+1,1} & \cdots & \phi_{m+1,m} & (-1)^{m+1} \end{pmatrix} \begin{pmatrix} \gamma_0 \\ \gamma_1 \\ \vdots \\ \gamma_m \\ \epsilon \end{pmatrix} = \begin{pmatrix} f(\lambda_0) \\ f(\lambda_1) \\ \vdots \\ f(\lambda_m) \\ f(\lambda_{m+1}) \end{pmatrix}, \qquad (5.4)$$

where $\phi_{i,j} = \phi(\lambda_j)$. We denote the coefficient matrix, the right hand side and the solution vector in the above equation by T, v and w respectively. After solving (5.4), the first $m+1$ components of w are used to assemble the next candidate for $p_m(\lambda)$ using (5.1). The assembled polynomial is evaluated at the all mesh points. Local extrema are searched to form the next set of reference points. The iteration continues until the magnitude of the polynomial becomes the same at all reference points. The construction of the linear system (5.4) ensures that there are at least $m+2$ points at which the error function alternates in sign. A careful description of the algorithm is given in Figure 5.5. Some implementation details are addressed below.

Remark 1 The initial choice of reference points is completely arbitrary. The Remez algorithm tends to self-correct a bad choice of reference points through the exchange process.

Remark 2 Some choices of reference points may lead to an ill-conditioned T at early stage. But this can be overcome by solving $Tw = v$ in a least squares sense. Again, the exchange process tends to eliminate the ill-conditioning as the iteration proceeds.

Remark 3 The search for local extrema is often done on the mesh points. If this is not sufficient (due to a coarse mesh), Newton's method may be used to produce a more accurate reference point set.

Algorithm: Remez

Input: A function $f(\lambda)$ continuous on $[\alpha, \beta]$.
Output: Coefficients γ_j, $(j = 0, 1, 2, ..., m)$ such that $p_n(\lambda) = \sum_{j=0}^{m} \gamma_j \phi_j(\lambda)$ is the minmax polynomial approximation to $f(\lambda)$ on $[\alpha, \beta]$.

1. converged = false;
2. Construct a fine mesh on the interval $[\alpha, \beta]$;
3. Choose an initial reference point set $R = \{\lambda_0, \lambda_1, ..., \lambda_{m+1}\}$ in $[\alpha, \beta]$;
4. **do**
  4.1. Evaluate $v_i = f(\lambda_i)$, for $i = 0, 1, ..., m+1$, and form $v = (v_i)$;
  4.2. Form the matrix $T = (\phi_{i,j})$, where $\phi_{i,j} = \phi_j(\lambda_i)$;
  4.3. Solve $Tw = v$, where $w = (\gamma_0, \gamma_1, ..., \gamma_n, \epsilon)^T$;
  4.4. Evaluate $p(\lambda) = \sum_{j=0}^{j=n} \gamma_j \phi_j(\lambda)$ at all mesh points;
  4.5. Find local extrema of the error function $e(\lambda) = f(\lambda) - p(\lambda)$ on the mesh, and form a new reference point set R;
  4.6. **if** $\left| \frac{\max_{\lambda \in R} |e(\lambda)|}{\min_{\lambda \in R} |e(\lambda)|} - 1 \right| < tolerence$ **then** converged = true; **endif**
5. **while** (not converged);

Figure 5.5 Remez Exchange Algorithm for constructing a minmax polynomial approximant to $f(\lambda)$.

Remark 4 The Remez algorithm can be easily modified to solve a weighted minmax approximation problem

$$\min_{\lambda\in(\alpha,\beta),p(\lambda)\in\mathcal{P}_m} \|f(\lambda)-p(\lambda)\|_w,$$

where $\|\cdot\|_w$ is defined by

$$\|f(\lambda)\|_w = \max_{\alpha\le\lambda\le\beta} |f(\lambda)w(\lambda)|.$$

5.4 Minmax Polynomial Acceleration

In this section, we will introduce a polynomial accelerant designed to approximate $\frac{1}{\lambda}$ in a (weighted) ∞-norm. The first part of the discussion focuses on applying the Remez algorithm directly to $\frac{1}{\lambda}$. The asymptotic behavior of the designed polynomial is illustrated. In Section 5.4.2, we introduce a simplified design procedure that gives a near minmax solution. Finally, we discuss how to construct a minmax polynomial that has a sharp derivative near zero.

5.4.1 Minmax Approximation to $\frac{1}{\lambda}$

Using the Remez algorithm, one can design polynomials that provide sufficient separation of the spectrum near μ. Of course, the approximation must exclude the singularity point $\lambda = 0$. For convenience, we restrict ourselves again to the problem of computing the lowest eigenvalues of a symmetric positive definite matrix. If the lower bound α of the spectrum is provided, one can solve the approximation problem on the interval $[\alpha,\beta]$, where β is the upper bound of the spectrum that is often easy to obtain. Otherwise one may construct a minmax polynomial approximation to

$$\psi(\lambda) = \frac{1}{\lambda+\sigma},$$

for some small $\sigma > 0$. An important feature of this type of polynomial accelerant is that asymptotically $p_m(\lambda)$ converges to $\psi(\lambda)$ as $m \to \infty$. This is shown in Figure 5.6 where two minmax polynomial approximants defined on the interval $[10^{-3}, 1]$ are plotted. The dash-dotted curve corresponds to a polynomial of degree 10, and the solid curve corresponds to a polynomial of degree 50. Since the minmax polynomial

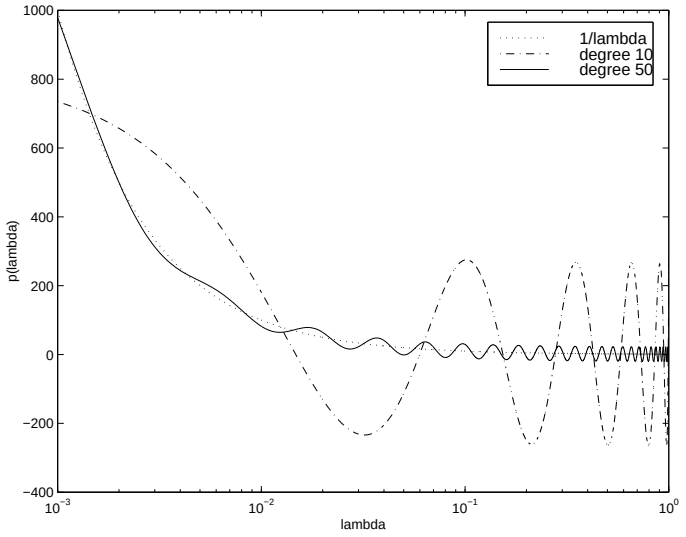

Figure 5.6 Minmax polynomial approximation to $\psi(\lambda) = 1/\lambda$ on the interval $[10^{-3}, 1]$. The dash-dotted curve corresponds to a 10-th degree polynomial constructed by the Remez algorithm. The solid curve corresponds to a 50-th degree polynomial. The dotted curve is the function $\psi(\lambda) = 1/\lambda$.

approximation provides the maximum error estimation, one can use this information to determine the appropriate degree of the approximating polynomial to be used in the eigenvalue calculation.

5.4.2 Near Minmax Approximation

We mentioned earlier that the initial choice of the reference points in the Remez algorithm is arbitrary. The algorithm tends to correct itself by picking out the correct

reference points along the way to convergence. However, in many cases, one may guess the location of the reference points, and further speed up the convergence by setting up a good starting configuration. In particular, a suitable choice of reference points consists of the local extrema of the shifted and scaled Chebyshev polynomial $C_{m+1}(\lambda;\alpha,\beta)$,

$$\lambda_j = \frac{\alpha+\beta-2\tilde{\lambda}_j}{\alpha-\beta}, \quad \text{where} \quad \tilde{\lambda}_j = \cos\Big[\frac{j\pi}{m+1}\Big], \quad j = 0,1,...,m+1.$$

This follows from the observation that the coefficients γ_j of the minmax approximation (5.1) decays rapidly as j gets larger. Thus, the error function must behave very much like the $m+1$st basis polynomial $C_{m+1}(\lambda;\alpha,\beta)$. By choosing the reference points at the extrema of $C_{m+1}(\lambda;\alpha,\beta)$ (without going through the process of Remez exchange) we are able to obtain a qualitatively good approximation to $\frac{1}{\lambda}$. The corresponding polynomial is sometimes referred to as a *near minmax* approximation [5, pp. 194-201]. Moreover, we can also avoid solving the linear system (5.4). This specific choice of reference points yields a T matrix of the form

$$T = \begin{pmatrix} 1 & 1 & \cdots & 1 & 1 \\ 1 & \cos\theta & \cdots & \cos m\theta & -1 \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ 1 & \cos m\theta & \cdots & \cos m^2\theta & (-1)^{m+1} \\ 1 & -1 & 1 & (-1)^{m+1} & (-1)^{m+2} \end{pmatrix},$$

where $\theta = \pi/(m+1)$. After scaling T by

$$\Sigma = \begin{pmatrix} 1/2 & & \\ & I_m & \\ & & 1/2 \end{pmatrix},$$

one can easily deduce, using the following trigonometric identities

$$1 + \cos 2\theta + \cos 4\theta + \cdots + \cos 2(m+1)\theta \quad = \quad 0,$$

$$1 + \cos^2\theta + \cos^2 2\theta + \cdots + \cos^2 m\theta \quad = \quad \frac{1+m}{2},$$

that the matrix $W = \Sigma T$ satisfies

$$W^2 = \frac{m+1}{2} I_{m+2}.$$

Thus the coefficients can be obtained by a few matrix vector multiplications. i.e.,

$$w = \frac{2}{m+1} \Sigma T \Sigma v.$$

We compare a 50-th degree polynomial constructed using this simple procedure with a 50-th degree minmax approximation in Figure 5.7. We observe that not only is the method of near minmax approximation simpler, it also renders a qualitatively better polynomial. On the region away from zero, the ripple-size of the near minmax polynomial is much smaller than its minmax counterpart.

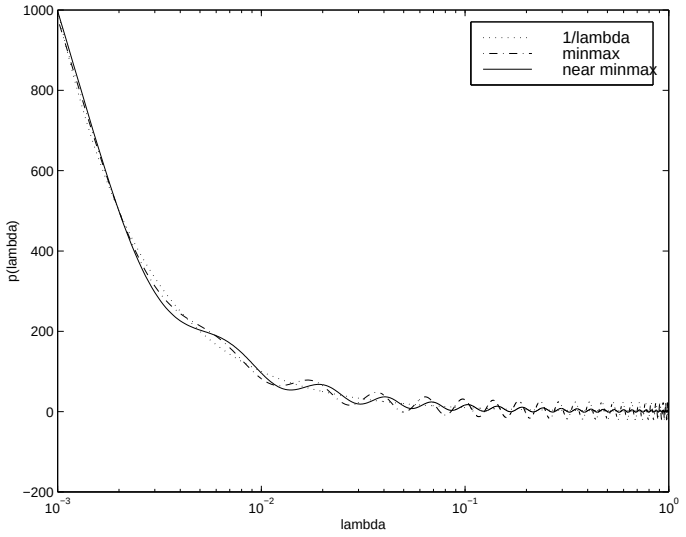

Figure 5.7 Approximation $1/\lambda$ by 50-th degree minmax and near minmax polynomials on $[10^{-3}, 1]$.

5.4.3 Constrained Minmax Approximation

All of the polynomials constructed above have large magnitudes near zero so that the smallest eigenvalues of A are transformed to the dominant eigenvalues of $p(A)$. In addition to this desirable property, we would also like the accelerating polynomial to be steep near zero. That is, we would like the derivatives of $p(\lambda)$ to be large in magnitude. This will help to separate those eigenvalues that are tightly clustered. In order to achieve this goal, we add some interpolative constraints to the approximation problem. For example, we might solve

$$\min_{\substack{p_m(\lambda) \in \mathcal{P}_m, \\ p_m^{(j)}(\omega) = \frac{(-1)^j}{\omega^{j+1}}}} \max_{\lambda \in [\alpha,\beta]} \|\frac{1}{\lambda} - p(\lambda)\|,$$

for some ω near α. (One could choose $\omega = \alpha$.) The constrained minmax approximation problem can be transformed into an unconstrained problem by writing the polynomial as

$$p_m(\lambda) = t_\ell(\lambda) + (\lambda - \omega)^\ell q_{m-\ell}(\lambda),$$

where $t_\ell(\lambda)$ consists the first $\ell + 1$ terms of the Taylor expansion of $\frac{1}{\lambda}$, and $q_{m-\ell}(\lambda)$ is to be determined. We expand $q_{m-\ell}$ in terms of scaled and shifted Chebyshev polynomials $C_j(\lambda; \alpha, \beta)$, and use the Remez algorithm to find the optimal expansion coefficient of $q_{m-\ell}(\lambda)$. After selecting a set of reference points, we enforce

$$\frac{1}{\lambda_j} - p_m(\lambda_j) = (-1)^j \epsilon,$$

for some unknown ϵ. This is equivalent to

$$q_{m-\ell}(\lambda_j) - \frac{(-1)^j \epsilon}{(\lambda_j - \omega)^\ell} = \frac{1}{(\lambda_j - \omega)^\ell \lambda_j}.$$

The unknown ϵ and the expansion coefficients γ_j's can be solved from

$$\begin{pmatrix} \phi_{0,0} & \cdots & \phi_{0,m-\ell} & h(\lambda_0) \\ \phi_{1,0} & \cdots & \phi_{1,m-\ell} & h(\lambda_1) \\ \vdots & \cdots & \vdots & \vdots \\ \phi_{m-\ell,0} & \cdots & \phi_{m-\ell,m-\ell} & h(\lambda_{m-\ell}) \\ \phi_{m-\ell+1,0} & \cdots & \phi_{m-\ell+1,m-\ell+1} & h(\lambda_{m-\ell+1}) \end{pmatrix} \begin{pmatrix} \gamma_0 \\ \gamma_1 \\ \vdots \\ \gamma_{m-\ell} \\ \epsilon \end{pmatrix} = \begin{pmatrix} r(\lambda_0) \\ r(\lambda_1) \\ \vdots \\ r(\lambda_{m-\ell}) \\ r(\lambda_{m-\ell+1}) \end{pmatrix},$$

where $\phi_{i,j} = C_j(\lambda_i; \alpha, \beta)$,

$$h(\lambda) = -\frac{(-1)^j}{(\lambda_j - \omega)^\ell} \quad \text{and} \quad r(\lambda) = \frac{t_\ell(\lambda) - \frac{1}{\lambda}}{(\lambda - \omega)^\ell}.$$

This formulation can also be viewed as a weighted minmax approximation with the weighting function $w(\lambda) = h(\lambda)$.

Intuitively, adding constraints to (5.3) causes some difficulty. It can be shown [3] that the maximum error produced by constrained minmax approximation is always greater than that produced by minmax approximation with no constraints. This is also demonstrated in Figure 5.8. We plotted both the constrained and unconstrained minmax polynomial of degree 50. Two constraints

$$p(\omega) = \frac{1}{\omega} \quad \text{and} \quad p'(\omega) = -\frac{1}{\omega^2}$$

are imposed in the constrained minmax approximation. Although the constrained minmax polynomial is steeper near ω than its unconstrained counterpart, it exhibits much larger error. Therefore, we shall only use the constrained minmax approximation when the gaps between the lowest eigenvalues are very small.

5.5 Least Square Approximation

Least squares approximation is a common technique used to minimize a weighted 2-norm of the error between a function and its approximant. We define a w-inner

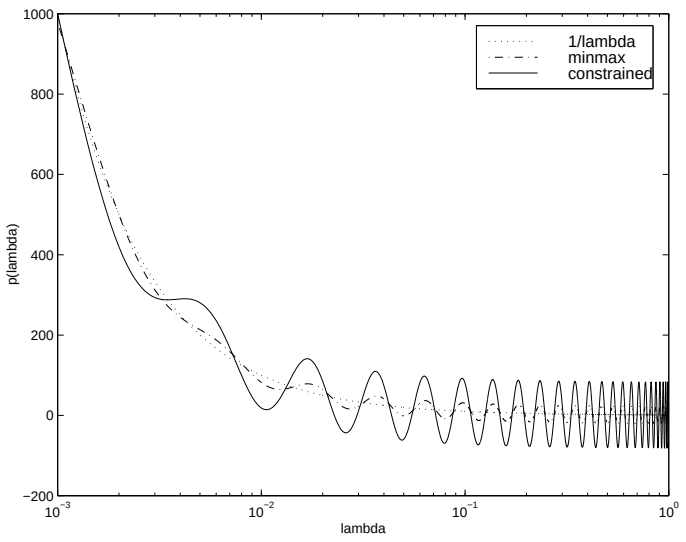

Figure 5.8 Constrained and unconstrained minmax polynomial approximation to $\psi(\lambda) = \frac{1}{\lambda}$ on the interval $[10^{-3}, 1]$. The solid curve corresponds to a 50-th degree constrained minmax polynomial. The dashed curve corresponds to a 50-th degree unconstrained minmax polynomial. The dotted curve is the function $\psi(\lambda) = \frac{1}{\lambda}$.

product $\langle \cdot, \cdot \rangle_w$ as:

$$\langle f, g \rangle_w = \int_\alpha^\beta f(\lambda) \cdot g(\lambda) w(\lambda) d\lambda,$$

where $w(\lambda)$ is a continuous function satisfying $w(\lambda) > 0$ on $[\alpha, \beta]$. The corresponding w-norm is

$$\|f\|_w = \sqrt{\int_\alpha^\beta f(\lambda)^2 w(\lambda) d\lambda}.$$

To accelerate the Lanczos process for computing the smallest eigenvalues of a positive definite matrix, we shall find the best polynomial $p_m(\lambda) \in \mathcal{P}_m$ such that

$$\|e(\lambda)\|_w = \|\frac{1}{\lambda} - p_m(\lambda)\|_w$$

is minimized. The least squares problem is closely related to the subject of orthogonal polynomials. Suppose $\{\phi_0(\lambda), \phi_1(\lambda), ..., \phi_m(\lambda)\}$ is an orthonormal basis of $\mathcal{P}_m$, i.e,

$$\langle \phi_i(\lambda), \phi_j(\lambda) \rangle_w = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

Then the least squares solution must have the form

$$p_m(\lambda) = \sum_{j=0}^{m} \gamma_j \phi_j(\lambda), \tag{5.5}$$

where the coefficients γ_j can be derived from the integral

$$\gamma_j = \langle \frac{1}{\lambda}, \phi_k(\lambda) \rangle_w = \int_\alpha^\beta \frac{\phi_j(\lambda)}{\lambda} w(\lambda) d\lambda.$$

It is well known that the orthonormal basis $\{\phi_0(\lambda), \phi_1(\lambda), ..., \phi_n(\lambda)\}$ may be generated via a three-term recurrence (*Stieltjes procedure* [86],)

$$\beta_{j+1}\phi_{j+1} = (\lambda - \alpha_j)\phi_j + \beta_j \phi_{j-1}, \quad \alpha_j = \langle \lambda\phi_j, \phi_j \rangle_w, \quad \beta_j = \langle \lambda\phi_j, \phi_{j-1} \rangle_w. \tag{5.6}$$

There are a variety choices for $w(\lambda)$. For example, the simplest is $w(\lambda) = 1$. This produces a set of $\{\phi_j(\lambda)\}$ known as the *Legendre* polynomial. If $\alpha = -1$ and $\beta = 1$, the three-term recurrence in (5.6) becomes

$$(j+1)\phi_{j+1}(\lambda) - (2j+1)\lambda\phi_j(\lambda) + j\phi_{j-1}(\lambda) = 0,$$

With $\phi_0(\lambda) = 1$ and $\phi_1(\lambda) = \lambda$, $\{\phi_j(\lambda)\}$ can be easily computed recursively. Another frequently used weight function is $w(\lambda) = 1/\sqrt{1-\lambda^2}$. With $\alpha = -1$ and $\beta = 1$, this $w(\lambda)$ leads to the well known Chebyshev basis whose recurrence formula

$$\phi_{k+1}(\lambda) = 2\lambda\phi_k(\lambda) - \phi_{k-1}(\lambda)$$

was shown in Section 2. In Figure 5.9, we compare a 10-th degree Chebyshev least squares polynomial defined on $[0.001, 1]$ with a Legendre least squares polynomial

of the same degree. It appears that the Chebyshev least squares polynomial drops faster than the Legendre least squares polynomial at the left end of the interval $[\alpha, \beta]$. However, its ripple-size is larger away from α. Therefore, the Chebyshev least squares polynomial is more suitable for separating eigenvalue clusters near α while the Legendre least squares polynomial is very effective in reducing the contribution of the unwanted eigencomponents of A to the Lanczos subspace.

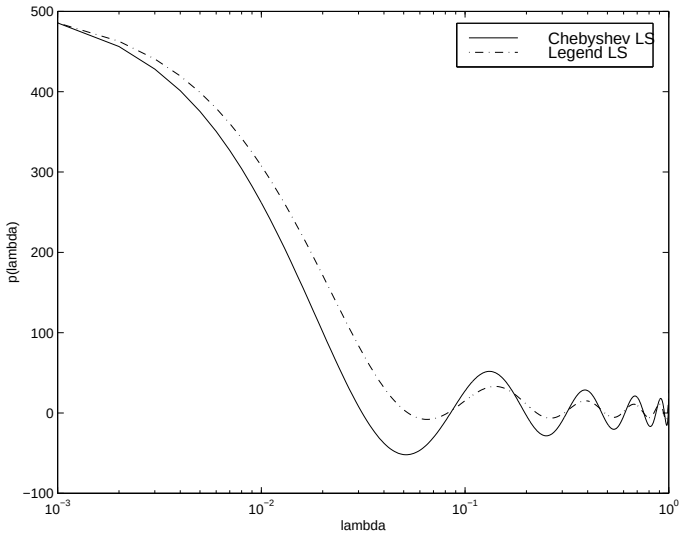

Figure 5.9 The solid curve corresponds to a 10-th degree Chebyshev least squares polynomial approximation to $\frac{1}{\lambda}$ constructed on $[0.001, 1]$. The dash-dot curve corresponds to a Legend least squares polynomial approximant of the same degree.

Just like the minmax approximation discussed earlier, one must prepare the lower and upper bound (α and β) of the spectrum for the least squares approximation discussed above. The least squares procedure minimizes the w-norm of the error function $\epsilon(\lambda)$ on $[\alpha, \beta]$. Although a polynomial designed in this manner converges to the ideal spectral transformation asymptotically (as $n \to \infty$), it is not optimal in the sense that a fair amount of work is done to minimized the approximation error on

subintervals that do not necessarily contain any eigenvalue. That is, such a design does not take into account the eigenvalue distribution of A. An ideal least squares approximation shall emphasize the minimization of the error on the discrete set of all eigenvalues.

Since eigenvalues of A are unknown, the best one can hope for is to devise a least squares design that takes advantage of an estimated eigenvalue distribution. As usual, the eigenvalue estimation may be accomplished by applying an m-step Lanczos to A directly. Given a starting vector v_0 that contains all eigencomponents of A, an m-step Lanczos procedure produces

$$AV_m = V_m T_m + f_m e_m^T, \quad V_m^T V_m = I_m, \quad V_m^T f_m = 0.$$

It is well known [89] that associated with the Lanczos process is a sequence of orthogonal polynomials $\{\phi_j\}$ whose three-term recurrence can be read off from the tridiagonal matrix

$$T_m = \begin{pmatrix} \alpha_1 & \beta_2 & 0 & \dots & 0 \\ \beta_2 & \alpha_2 & \ddots & & \vdots \\ & \ddots & \ddots & \ddots & \\ \vdots & & \ddots & \alpha_{m-1} & \beta_m \\ 0 & \dots & & \beta_m & \alpha_m \end{pmatrix}.$$

These polynomials are orthogonal with respect to a discrete inner product completely determined by the eigenvalues and eigenvectors of T_m.

Let $T_m = SDS^T$ be an eigendecomposition of T_m, where $D = \mathrm{diag}(\theta_1, \theta_2 \cdots \theta_m)$. It can be verified [57, pp. 125-129] that

$$S = \begin{pmatrix} \phi_0(\theta_1) & \phi_0(\theta_2) & \cdots & \phi_0(\theta_m) \\ \phi_1(\theta_1) & \phi_1(\theta_2) & \cdots & \phi_1(\theta_m) \\ \vdots & \vdots & \cdots & \vdots \\ \phi_{m-1}(\theta_1) & \phi_{m-1}(\theta_2) & \cdots & \phi_{m-1}(\theta_m) \end{pmatrix}.$$

The orthogonal polynomials generated by the Lanczos process are orthogonal with respect to the inner product

$$\langle f, g \rangle_\tau = \sum_{j=1}^{m} f(\theta_j) g(\theta_j) \tau_j^2,$$

where $\tau_j = \phi_0(\theta_j)$.

Using this inner product and the orthogonal polynomials $\{\phi_j(\lambda)\}$, we can solve the least squares problem by evaluating the coefficients in (5.5) using

$$\gamma_j = \langle \frac{1}{\lambda}, \phi_j \rangle_\tau = \sum_{i=1}^{m} \frac{\phi_j(\theta_i)}{\theta_i} \tau_i^2.$$

We will call the polynomial constructed in this manner a *Ritz polynomial.* We mention in passing that this polynomial is equivalent to the polynomial produced implicitly by the conjugate gradient algorithm for solving $Ax = b$.

To give an example, we apply a 10-step Lanczos run to the 100×100 tridiagonal matrix with 2 on the diagonal and -1 on the super and sub-diagonals. The corresponding least squares polynomial approximation to $\frac{1}{\lambda}$ is shown in Figure 5.10. The circles in the graph correspond to transformed eigenvalues of A. In Figure 5.11, we demonstrate the asymptotic behavior of such a least squares design by plotting both a 50-th degree least squares polynomial and the corresponding error function $e(\lambda) = \frac{1}{\lambda} - p_{50}(\lambda)$. It is seen that as more Ritz values θ_j converge to eigenvalues of A, the least squares polynomial indeed minimizes error on the spectrum of A.

5.6 Polynomial Accelerants for Computing Interior Eigenvalues

Many of the techniques introduced above can be adopted to construct polynomial accelerants for interior eigenvalue calculation. Suppose eigenvalues near μ are of interest, an ideal polynomial must have large magnitude near the target shift $\lambda = \mu$ and small magnitude elsewhere. We will refer to this type of polynomial as the

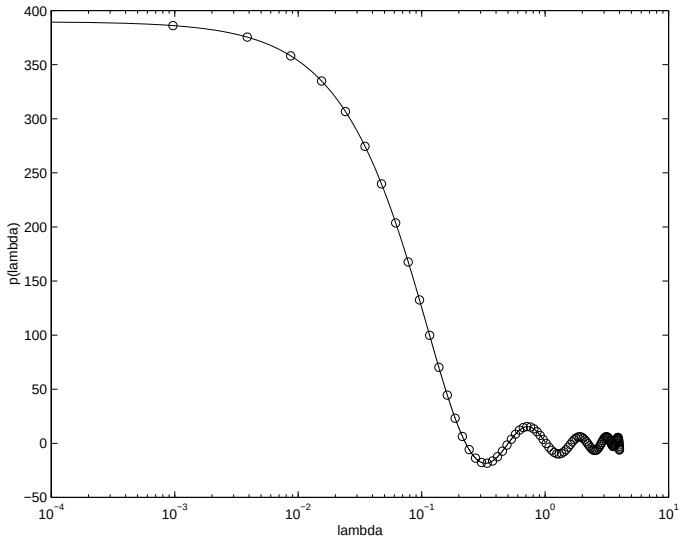

Figure 5.10 The solid curve corresponds to a 10-th degree Ritz least squares polynomial implicitly constructed by applying a 10-step Lanczos run to A. The transformed eigenvalues are plotted in circles.

bandpass polynomial. Following the terminology used in digital filter design, we will call the interval that contains only the wanted eigenvalues a *passband*, and the interval that does not contain desired eigenvalues a *stopband*. The ratio between the passband and stopband magnitude and the width of the passband determines the quality of the constructed polynomial.

5.6.1 Chebyshev and Kernel Polynomials

If a sequence of orthogonal polynomials $\{\phi_j(\lambda)\}$ are generated on a interval $[\alpha, \beta]$ that contains the point $\lambda = \mu$, the Kernel polynomial $K(\lambda; \mu)$ corresponding to $\{\phi_j(\lambda)\}$ yields the graph shown in Figure 5.12. The polynomial indeed has large magnitude at the shift $\lambda = \mu$, and its relative extrema decrease monotonically as λ moves away from μ. A similar but different polynomial can be built from a Chebyshev polynomial alone. Since the Chebyshev polynomial $T_m(\lambda; \alpha, \beta)$ has large magnitude outside of

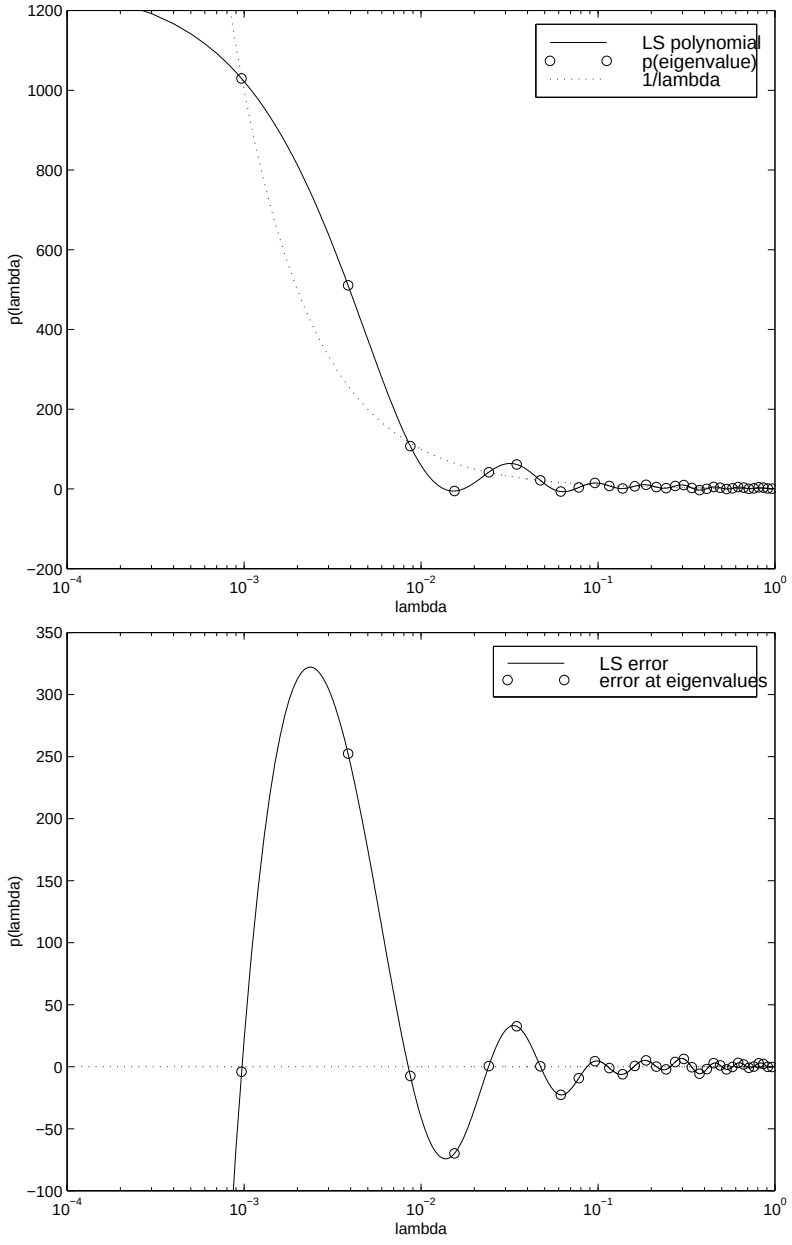

Figure 5.11 The solid curve shown in the top figure corresponds to a 50-th degree Ritz least squares polynomial. The circles in that figure indicate the location of the transformed eigenvalues. The dotted curve corresponds to the rational function $\frac{1}{\lambda}$. The solid curve in the bottom figure shows the absolute error $e(\lambda) = \frac{1}{\lambda} - p_m(\lambda)$ of the approximation.

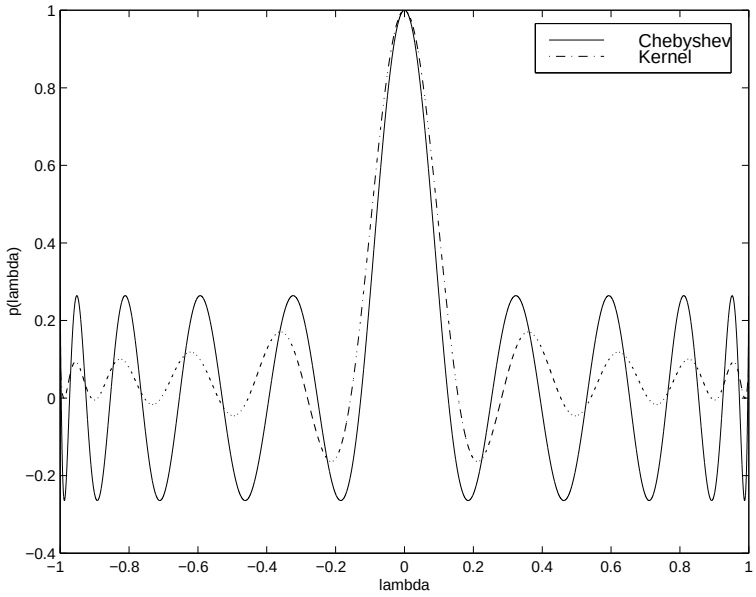

Figure 5.12 The solid curve corresponds to a 20-th degree Chebyshev polynomial constructed with $\alpha = 0.01$ and $\beta = 1$. The dash-dotted curve corresponds to a Kernel polynomial $K(\lambda; 0)$ constructed using with a Chebyshev basis $\{T_j(\lambda; -1, 1)\}$.

the interval $[\alpha, \beta]$, the polynomial

$$p_{2m}(\lambda) = T_m\Big((\lambda - \mu)^2; \alpha, \beta\Big)$$

must have large magnitude near μ. It is bounded uniformly by 1 on $[(\mu-\beta)^2, (\mu-\alpha)^2]$ and $[(\mu + \alpha)^2, (\mu - \beta)^2]$. A 20-th degree polynomial designed by this method is compared with a 20-th degree Kernel polynomial in Figure 5.12.

5.6.2 Minmax Polynomials

Polynomials that bump up at μ can also be constructed by the minmax approximation procedure. One can write the polynomial to be constructed as

$$p_m(\lambda) = 1 + (\lambda - \mu)^2 q_{m-2}(\lambda),$$

where $q_{m-2}(\lambda)$ is to be found iteratively by the Remez algorithm. There are two ways to compute q_{m-2}. A direct approach is to specify two cutoff points ω_1 and ω_2 such that the magnitude of $p_m(\lambda)$ is minimized within $[\alpha, \omega_1]$ and $[\omega_2, \beta]$. This is equivalent to the Grcar residual polynomial defined in [26, 3]. In this case, the intervals $[\alpha, \omega_1]$ and $[\omega_2, \beta]$ are treated as *stopbands* and the interval $[\omega_1, \omega_2]$ is treated as a *passband.*

The second way of constructing $q_{m-2}(\lambda)$ is to prescribe the maximum magnitude of the stopband, and use a Remez-like algorithm to minimize the bandwidth of the passband. This technique is successfully used in digital filter design [78], and is incorporated in SPEIG - a MATLAB implementation of the implicitly restarted Arnoldi method [61]. (SPEIG has evolved into `EIGS` in MATLAB 5.1.) In the Lanczos setting, the second approach is preferred because it is much easier to specify the magnitude bound of the stopband than estimating optimal cutoff points μ_1 and μ_2. Figure 5.13 shows two 10-th degree bandpass polynomials constructed on the interval $[-1, 1]$ and $[-1, 10]$ respectively. They are compared with $q_{10} = T_5(\lambda^2)$ where $T_5(\lambda)$ is a shifted

and scaled Chebyshev polynomial. In the symmetric case (the interval $[-1,1]$ being symmetric about zero), these two polynomials are almost identical. When the interval is not symmetric about 0, the bandpass polynomial designed by a minmax procedure produces a much narrower passband.

5.6.3 Least Square Polynomials

Saad proposed using least squares techniques to construct a bandpass polynomial $p_m(\lambda) = 1 - \lambda q_{m-1}(\lambda)$ that has the least ℓ_2-norm [72]. The crux of this type of least squares approximation is the generation of polynomials that are orthogonal over two disjoint intervals $[\alpha_1, \beta_1]$ and $[\alpha_2, \beta_2]$. Given two weight functions $w_1(\lambda)$ and $w_2(\lambda)$, the inner product associated with these polynomials is defined as

$$\begin{aligned} \langle f, g \rangle &= \int_{\alpha_1}^{\beta_1} f g w_1(\lambda) d\lambda + \int_{\alpha_2}^{\beta_2} f g w_2(\lambda) d\lambda \\ &= \langle f, g \rangle_1 + \langle f, g \rangle_2. \end{aligned}$$

In [72], Saad presented a mechanism for generating orthogonal polynomials associated with two Chebyshev weights

$$w_1(\lambda) = \frac{2}{\pi} \frac{1}{\sqrt{\beta_1^2 - (\lambda - \alpha_1)^2}}, \quad w_2(\lambda) = \frac{2}{\pi} \frac{1}{\sqrt{\beta_2^2 - (\lambda - \alpha_2)^2}}.$$

A general procedure for generating orthogonal polynomials on several disjoint intervals is described in [19] and [20].

5.7 Numerical Examples

We apply polynomial transformations presented earlier to two particular eigenvalue problems. The first problem was shown before in Section 3.4.3. Eigenvalues of a reactive scattering Schrödinger operator are of interest. The matrix representation of the operator is positive definite but very ill-conditioned. Smallest eigenvalues

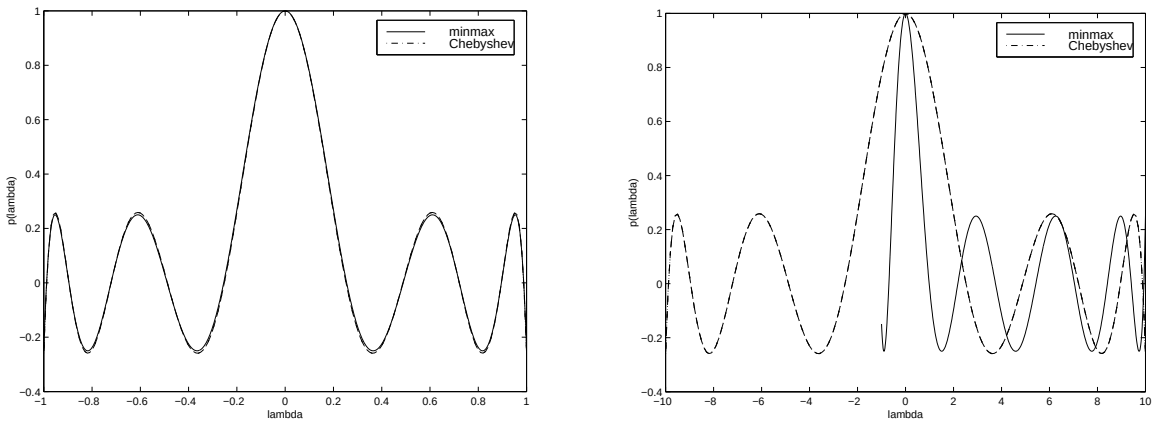

Figure 5.13 Symmetric and Non-symmetric Bandpass polynomials. The polynomials shown in the left graph are define on an interval that is symmetric about zero. The polynomial generated by a direct Remez procedure gives a narrow passband bandwidth on a non-symmetric interval.

are difficult to obtain because the spectrum of A contains many uninteresting but dominant eigenvalues. Our second example is related to the analysis of metal insulator transition in disordered material (the Anderson Model) [48, 1]. The corresponding matrix is indefinite. Eigenvalues nearest to zero are of interest.

These problems are solved using `ARPACK` [40]. All computations described in this section are performed on a SUN Ultra2 in double precision. The convergence of a Ritz pair (θ_j, y_j) is declared when the relative Ritz error estimate falls below a tolerance of 10^{-8}.

Example 1

The matrix A we choose to experiment with here is of dimension 1024×1024. The problem is not particularly large, hence we can find the smallest eigenvalues of A by running `ARPACK` directly on A although this is less efficient. We will refer to this calculation as the *direct* calculation in the following discussion. When the dimension of

the problem becomes larger, it will be more difficult to obtain the smallest eigenvalues through the direct computation.

From the direct calculation, we observe that smallest eigenvalue of A is $\lambda_1 = 3.4234 \times 10^4$. It has multiplicity 2. The largest eigenvalue is $\lambda_{1024} = 2.5369 \times 10^{10}$.

In Table 5.1, we compare the performance of various polynomial spectral transformations $p_m(\lambda)$ by counting the number of matrix vector multiplications and CPU time used by `ARPACK` when it is applied to $p_m(A)$. All polynomials constructed have degree 10. Of course, $p_{10}(A)$ is never explicitly formed. We supply only the action of the matrix-vector multiplication $w \leftarrow p_{10}(A)v$. We seek 10 smallest eigenvalues, and set the dimension of the Krylov subspace to be 40.

polynomial	MATVECs	CPU time (sec)
Minmax	5240	32.4
Chebyshev	5230	35.6
Leja	6790	41.8
Kernel	6760	42.0
Chebyshev least squares	6770	42.2
Ritz least squares	6100	42.4
Near Minmax	6770	42.8
Legendre least squares	7290	50.8
Direct (Identity)	7392	78.9

Table 5.1 Comparison of polynomial transformations

The Chebyshev polynomial requires two cutoff parameters α and β as discussed in Section 5.1. We choose $\alpha = 3.4 \times 10^7$ and $\beta = 2.5 \times 10^{10}$. The Kernel polynomial is not as sensitive to the value of α as the Chebyshev polynomial. Typically, one can choose α value near the low end of the spectrum. We set $\alpha = 3.4 \times 10^5$ in our experiment. The Minmax, Near minmax, Chebyshev and Legendre least squares polynomial all require estimations of the upper (β) and lower (α) bound of the spectrum. In the

experiment, we use $\alpha = 3.4 \times 10^4$ and $\beta = 2.54 \times 10^{10}$. The Ritz least squares polynomial is a parameter free polynomial. The graphs of all these polynomials are drawn in Figure 5.14.

polynomial	MATVECs	CPU time (sec)
Chebyshev	89,180	1677
Kernel	155,600	2640
Minmax	117,880	1842

Table 5.2 Comparison of bandpass polynomial transformations

From Table 5.1, we observe that all polynomial accelerated Lanczos calculations perform better than the direct calculation. The minmax approximation appears to be the best among all polynomials constructed in this experiment. It reduces CPU time of the direct Lanczos calculation by a factor of two. The parameter-free polynomial acceleration based on the Ritz least squares approximation is almost as good as the minmax polynomial transformation.

Example 2

The Anderson model is developed to describe a quantum-mechanical electron in a crystal with impurities [34]. Eigenvalues of the Schrödinger operator

$$H_\sigma = \frac{1}{2}\Delta + \sigma V,$$

which consists of both a Hamiltonian Δ and a random potential term V, are of interest. The parameter σ is called the *disorder*. The spectrum of H_σ is known to include both positive and negative eigenvalues. The computations presented below are performed on a matrix obtained from a discretization of a 3-dimensional H_σ [48]. The dimension of the matrix is $15,626 \times 15,625$. Using direct calculation, we observe

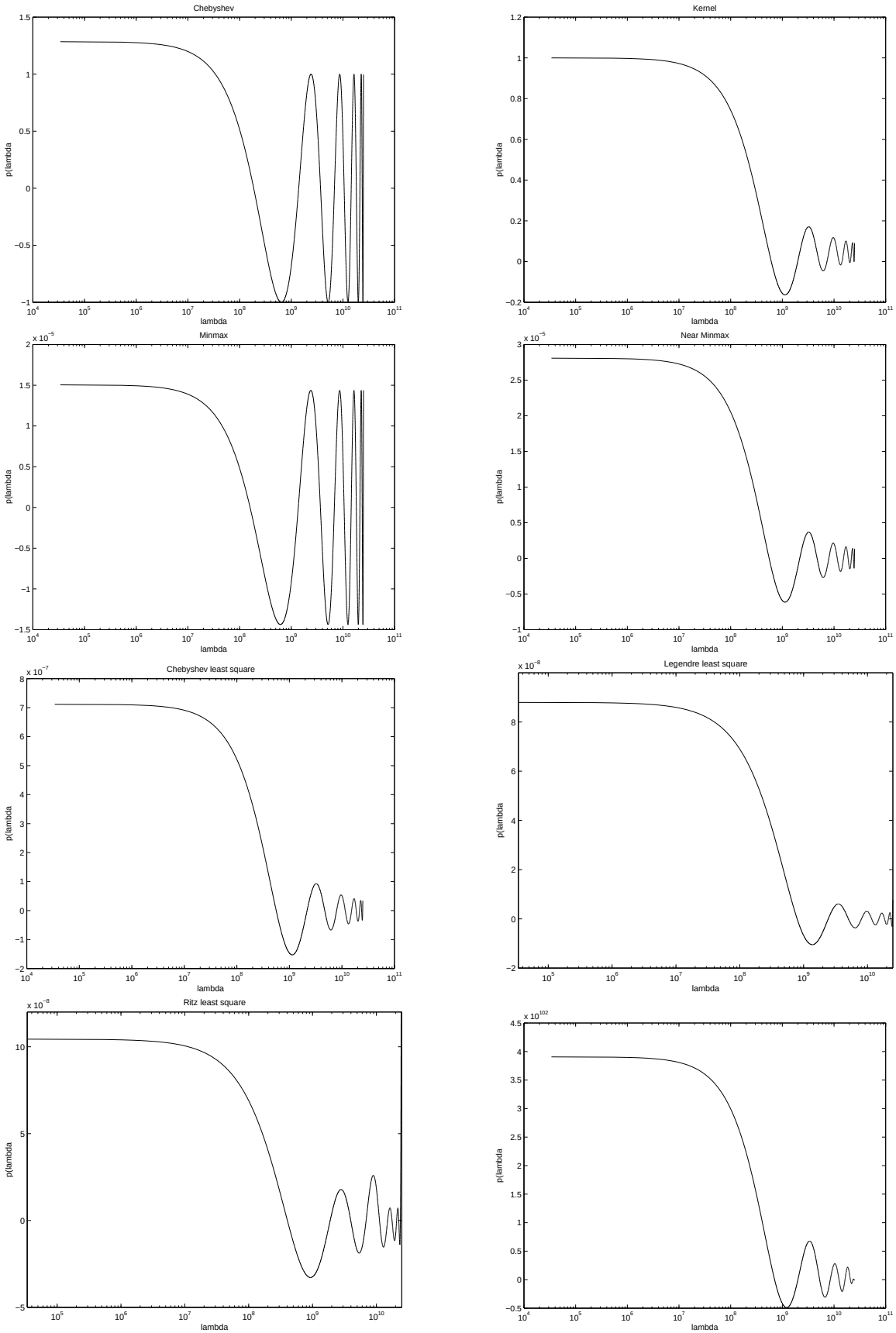

Figure 5.14 Comparison of various polynomial transformations

that the algebraically smallest and largest eigenvalues of A are

$$\lambda_1 = -18.14, \quad \lambda_{15625} = 18.51$$

respectively. The structure of the matrix is shown in Figure 5.15. Just like the reactive scattering example, a sparse factorization tends to destroy the sparsity of the matrix A.

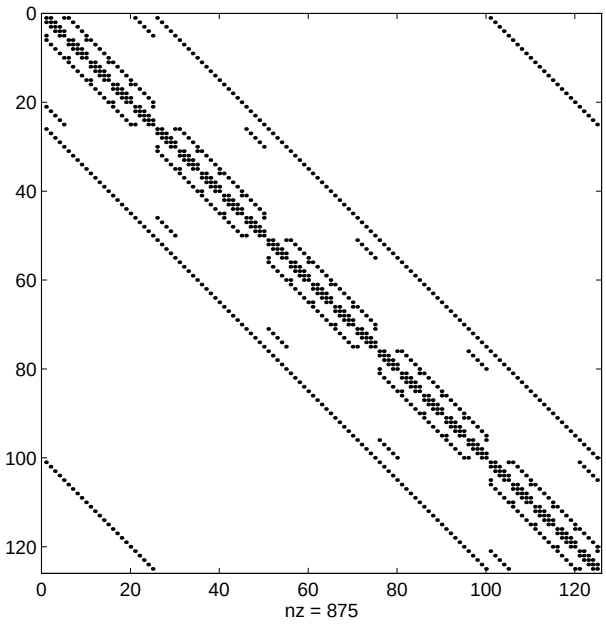

Figure 5.15 The sparsity pattern of the Anderson matrix.

In Table 5.2, we compare the performance of the Chebyshev, Minmax and Kernel polynomial by listing the total number of matrix vector multiplications and CPU time required by each method. The corresponding polynomials are plotted in Figure 5.16. All polynomials constructed are of degree 20. We seek 10 eigenvalues nearest to 0, and set the dimension of the Krylov subspace to be 100.

The computed eigenvalues are listed in Table 5.3.

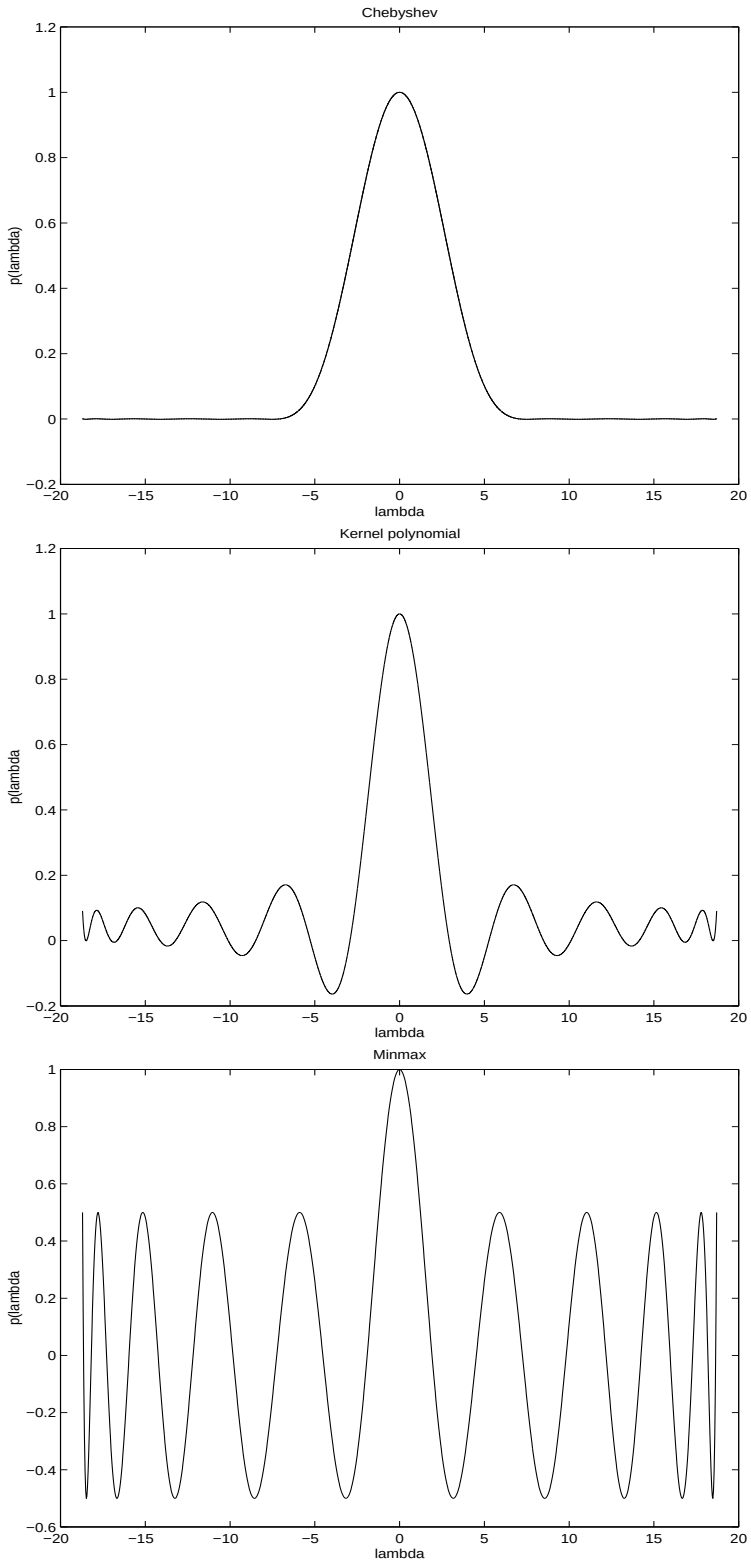

Figure 5.16 Comparison of various polynomial transformations applied to the Anderson model eigenvalue calculation.

λ_1	-9.2371×10^{-3}
λ_2	-3.2661×10^{-3}
λ_3	5.1842×10^{-4}
λ_4	1.8468×10^{-3}
λ_5	2.7786×10^{-3}
λ_6	3.4986×10^{-3}
λ_7	5.6559×10^{-3}
λ_8	6.2122×10^{-3}
λ_9	6.9863×10^{-3}
λ_{10}	1.0086×10^{-2}

Table 5.3 Computed eigenvalues of the Anderson model

It appears that the Chebyshev polynomial gives the best performance. But it does require some nontrivial estimation of the cutoff parameters α and β. In the experiment, we choose $\alpha = 7.0$ and $\beta = 18.75$. The Kernel and Minmax polynomials require slightly more matrix vector multiplications. However, they only require the lower and upper bound of the spectrum which are readily available from a direct Lanczos calculation.

In this section, we have provided a survey on techniques for constructing polynomial accelerants for the Lanczos iteration. Most of these techniques are not new. Some are well known in the approximation theory literature. However, they have not been used extensively in the eigenvalue computation. The previous numerical examples have demonstrated that polynomial transformation can be very effective in capturing the clustered and interior eigenvalues.

The construction of these polynomials often requires some knowledge of the spectrum a priori. This information can usually be provided by applying a few steps of Lanczos directly to A. However, a few questions remain as to which polynomial one

should use for a particular problem and how to set the degree of the polynomial to achieve optimal performance.

We have only considered symmetric eigenvalue problems here. Constructing polynomial spectral transformations for nonsymmetric eigenvalue problems is more challenging. In particular, the idea of approximating $\frac{1}{\lambda}$ by some polynomial does not extend to the complex plane because the maximum value of any analytic function cannot occur in the interior of any domain. A commonly used technique is to define an ellipse that encloses most of the unwanted eigenvalues and construct a shifted and scaled Chebyshev polynomial that is bounded inside the ellipse and increases rapidly (in magnitude) outside that region [43, 73, 29, 77].

We shall emphasize that although polynomial spectral transformation is a powerful tool for accelerating the Arnoldi/Lanczos iteration, it should only be used when factoring $A - \sigma I$ is prohibitively expensive or when there is no obvious preconditioner for the linear system $(A - \sigma I)x = b$. If those are not the cases, one should consider using rational transformations discussed in the last chapter, the rational restarting technique through the TRQ iteration, or the Jacobi Davidson algorithm to be discussed in the next Chapter.

Chapter 6

Newton-like Acceleration

A number of people [88, 65, 11, 60, 87] have treated the algebraic eigenvalue problem as a problem of solving the nonlinear equation

$$Ax - \lambda(x)x = 0, \tag{6.1}$$

where $\lambda(x) = \frac{x^H Ax}{x^H x}$. Under this framework, a Newton-like of method may be applied to find the zeros of (6.1). The recently proposed Jacobi-Davidson (JD) method [79] can be motivated by this approach. Since Newton's method is only locally convergent, a global search strategy must be developed to provide a reasonable starting guess. In JD, this is often accomplished by a k-step Arnoldi iteration. Therefore, in some sense, JD can be viewed as a means to accelerate the Arnoldi method. However, we wish to emphasize that the JD algorithm itself does not belong to the family of Krylov subspace methods. It is also important to note that the updating strategy used in JD is slightly different from the traditional Newton update in which the Newton correction is added directly to the previous approximation. We will come back to this observation in the Section 6.2. In Section 6.1, we derive the basic JD iteration for computing a single eigenpair. This is followed by the discussion of a more general approach that is capable of producing an orthonormal basis of an invariant subspace of A. In either case, one must solve a linear system to obtain a Newton correction vector. Section 6.3 addresses issues related to solving this linear system, especially when a preconditioned iteration solver is used. A numerical example is provided in Section 6.4 to compare the performance of JD with the inexact TRQ algorithm presented in Chapter 3.

6.1 Basic Formulation

6.1.1 Eigenvector Correction

Suppose θ is an approximation to a desired eigenvalue and u is the corresponding eigenvector approximation with unit length. They may, for example, be a Ritz pair obtained from a k-step Arnoldi iteration. We can improve the approximation by seeking a correction pair (γ, z) such that

$$A(u+z) - (\theta+\gamma)(u+z) = 0 \quad \text{and} \quad u^H z = 0. \tag{6.2}$$

If u is sufficiently close to x, we drop the second order term γz, and solve

$$(A-\theta I)z - \gamma u = -r \quad \text{and} \quad u^H z = 0, \tag{6.3}$$

where $r = Au - \theta u$. The above equations may also be expressed in the following *bordered* form

$$\begin{pmatrix} A-\theta I & u \\ u^H & 0 \end{pmatrix} \begin{pmatrix} z \\ -\gamma \end{pmatrix} = \begin{pmatrix} -r \\ 0 \end{pmatrix}. \tag{6.4}$$

If $A-\theta I$ is nonsingular (i.e., θ has not converged,) we may perform a block Gauss elimination to obtain

$$\begin{pmatrix} I & 0 \\ u^H(A-\mu I)^{-1} & 1 \end{pmatrix} \begin{pmatrix} A-\theta I & u \\ 0 & u^H(A-\theta)^{-1}u \end{pmatrix} \begin{pmatrix} z \\ -\gamma \end{pmatrix} = \begin{pmatrix} -r \\ 0 \end{pmatrix}.$$

Back substitution yields the correction vector

$$\begin{aligned} z &= \gamma(A-\theta I)^{-1}u - (A-\theta I)^{-1}r & (6.5)\\ &= \gamma(A-\theta I)^{-1}u - u, & (6.6) \end{aligned}$$

where

$$\gamma = \frac{u^H(A-\theta I)^{-1}r}{u^H(A-\theta I)^{-1}u} = \frac{1}{u^H(A-\theta I)^{-1}u}.$$

After updating u by adding the correction z, we have

$$\hat{u} = u + z = \gamma(A - \theta I)^{-1}u, \tag{6.7}$$

a vector that appears in an inverse iteration.

As pointed out in [79], the idea of eigenvector correction dates back to 1845 when Jacobi considered a special correction scheme for improving an approximate eigenpair of a diagonally dominant matrix [31]. (Of course, Jacobi did not formulate the problem in modern linear algebra parlance since the notation of matrix was not available then.) The JD algorithm to be developed below was motivated in part by his correction scheme.

The connection between the inverse iteration and the method of eigenvector correction was expounded in [60]. However the emphasis there was on how to compute the inverse iteration more accurately.

Notice that we did not follow the standard definition of Newton's method in deriving the correction vector. However, the method described is essentially a Newton iteration in the sense that equation (6.3) is a first order approximation to (6.2). If θ is taken to be the Rayleigh quotient $u^H Au$, it is not difficult to show that the standard definition of Newton's method gives rise to the same equation as given in (6.4).

If $\theta = u^H Au$, it is sometimes convenient to rewrite the bordered system (6.4) in the following form:

$$(I - uu^H)(A - \theta I)(I - uu^H)z = -r. \tag{6.8}$$

This equation can be derived directly from (6.3) by multiplying $I - uu^H$ from the left and using the fact

$$u^H r = 0 \quad \text{and} \quad u^H z = 0.$$

The advantage of this representation is that it reveals the relationship between z and r under the mapping

$$A^{\theta}_{u^{\perp}} : u^{\perp} \mapsto u^{\perp},$$

where $u^{\perp}$ denotes the orthogonal complement of u, and $A^{\theta}_{u^{\perp}}$ is defined as

$$A^{\theta}_{u^{\perp}} \equiv (I - uu^H)(A - \theta I)(I - uu^H).$$

A Krylov subspace solver applied to (6.8) (with a starting vector that is orthogonal to u) provides an approximation that lies in the subspace $u^{\perp}$ whereas a Krylov subspace generated by the border matrix

$$B = \begin{pmatrix} A - \theta I & u \\ u^H & 0 \end{pmatrix} \tag{6.9}$$

lacks this consistency property. That is, if $(\hat{z}, \hat{\gamma})^T$ is an approximation produced by applying a Krylov subspace solver to (6.9), the vector $\hat{z}$ is unlikely to be in $u^{\perp}$. We will reiterate this observation in Section 6.3.

Notice the structural similarity between (6.4) and (3.6), and also between (6.8) and (3.17). This similarity is even more pronounced in Section 6.2 where the vector u is replaced by an $n \times k$ matrix.

We note that the orthogonality condition $u^H z = 0$ used in (6.2) is not unique. In principle, one can impose $\tilde{u}^H z = 0$ for any $\tilde{u}$ not orthogonal to z. Thus, the most general form of (6.4) is:

$$\begin{pmatrix} A - \theta I & u \\ \tilde{u}^H & 0 \end{pmatrix} \begin{pmatrix} z \\ -\gamma \end{pmatrix} = \begin{pmatrix} -r \\ 0 \end{pmatrix}.$$

However, in practice, $\tilde{u} = u$ seems to be the most natural and convenient choice.

If, in addition, one computes an approximation eigenvalue from the *Petrov-Galerkin* criterion:

$$w^H (A - \theta I) u = 0$$

to obtain $\theta = \frac{w^H A u}{w^H u}$, it follows from (6.3) that

$$\Big(I - \frac{u w^H}{w^H u}\Big)(A - \theta I)\Big(I - \frac{u \tilde{u}}{\tilde{u} u^H}\Big) z = -r.$$

This is the most general form of (6.8).

6.1.2 Subspace Augmentation

It will be less exciting if we update u by simply adding the correction vector z to arrive at an inverse iteration. A more interesting approach taken by Davidson in [15] is to normalize z and make it a basis vector of an augmented subspace $\mathcal{S} = \text{span}\{u, z\}$ from which new eigenvalue and eigenvector approximations are extracted. If we set $V = (u, z)$, the new approximating pair can be computed by imposing the Galerkin condition:

$$V^H(AVy - \theta Vy) = 0.$$

Notice that in this scheme u and z are combined in a "best possible" way to provide an improved approximation to the desired eigenvector. We will later show that this mechanism also allows some flexibility in computing z since a Newton update is not strictly enforced. All one cares to do is to augment the previous subspace with a vector pointing roughly in the direction of a Newton correction.

If the new approximation is still not accurate enough, we repeat the correction process and use the correction vector z to further expand $\mathcal{S}$ after orthogonalizing it against V. The advantage of JD over a single vector inverse iteration is apparent: Besides providing an approximation to a single eigenpair of A, the JD iteration also produces a subspace from which approximations to other eigenpairs can be obtained. We will postpone the discussion on how to compute several eigenpairs until Section 6.2. A simple version of the JD algorithm is outlined in Figure 6.1.

Algorithm: Jacobi Davidson

Input: a matrix A, a starting vector v_0
Output: eigenpair approximation (θ, u) such that $\|Au - \theta u\|$ is small.

1. $v \leftarrow v_0/\|v_0\|$, $\theta \leftarrow v^H Av$, $u \leftarrow v$; converged $\leftarrow$ FALSE;
2. $V \leftarrow (v)$; $H \leftarrow (\theta)$;
3. while not converged **do**
  3.1 for $j = 1, 2, 3, ..., m-1$
    3.1.1 Solve (approximately)

$$\begin{pmatrix} A - \theta I & u \\ u^H & 0 \end{pmatrix} \begin{pmatrix} z \\ -\gamma \end{pmatrix} = \begin{pmatrix} -r \\ 0 \end{pmatrix};$$

    3.1.2 $z \leftarrow (I - VV^H)z$;
    3.1.3 $V \leftarrow (V, z)$; $H \leftarrow V^H AV$;
    3.1.4 Compute all eigenpairs of H, and select a desired pair (θ, s);
    3.1.5 Put $u \leftarrow Vs$;
    3.1.6 if $\|Au - \theta u\| <$ some tolerance, converged $\leftarrow$ TRUE;
  3.2 end for;
  3.3 if (not converged) Restart: put $V \leftarrow (u)$, $H \leftarrow (\theta)$; go to 3; **end if**;
4. end while;

Figure 6.1 The Jacobi-Davidson Iteration.

6.1.3 Inexact Newton Correction

A number of researchers have analyzed the convergence rate of Newton's method when the Newton step is computed approximately by an iterative solver. Under some appropriate assumptions on the error associated with the approximate Newton correction, one can show that the *inexact* Newton method converges super-linearly.

The idea of inexact Newton calculation fits naturally in the JD algorithm. Although no rigorous analysis has been done to characterize the convergence, numerical evidence has been provided in [79] to indicate the success of this approach.

In [79], Sleijpen and Van der Vorst proposed taking

$$z = \gamma M^{-1}u - M^{-1}r, \quad \text{where} \quad \gamma = \frac{u^H M^{-1} r}{u^H M^{-1} u}. \tag{6.10}$$

The matrix M^{-1} shall be viewed as an approximate inverse of $A-\theta I$, and the formula for the coefficient γ follows directly from the condition $u^H z = 0$. It is assumed that M^{-1} is much easier to compute than $(A-\theta I)^{-1}$. In practice, $M^{-1}r$ and $M^{-1}u$ are typically computed by solving $Mx = r$ and $Mx = u$ iteratively. It is easy to see that Equation (6.10) can be deduced from (6.5) by simply replacing $(A-\theta I)^{-1}$ with M^{-1}. While it is possible to use (6.6), which yields a correction vector

$$z = \gamma M^{-1}u - u, \quad \text{where} \quad \gamma = \frac{1}{u^H M^{-1} u},$$

it is reported in [79] that this vector may lie in almost the same direction as u. Thus computing an orthogonal correction $z \leftarrow (I - VV^H)z$ may be difficult in floating point arithmetic.

It is interesting to note that the original Davidson's method can be derived from (6.10) by taking M to be the the diagonal of $A - \theta I$ and setting $\gamma = 0$. The Olsen method [53] can be derived in a similar way. Numerical experiments are shown in [79] to demonstrate that the JD algorithm is superior to Davidson's method, especially when A is not diagonally dominant.

6.1.4 Termination and Restart

If the correction equation (6.4) is solved exactly, the subspace generated by the JD algorithm

$$\mathcal{S} = \{v_0, v_1, ..., v_k\}, \quad \text{where} \quad v_j = (A - \theta_j I)^{-1} v_{j-1}$$

is equivalent to (4.3). Since $\mathcal{S}$ is expanded with Newton's directions, the JD iteration converges rapidly (typically cubic for symmetric problems and quadratic for non-symmetric problems.) Therefore, the dimension of the subspace required to provide eigenvalue and eigenvector approximation is often small. However, if (6.4) is solved with some error, one may need a larger subspace to assemble an accurate eigen-approximation. To avoid expanding the subspace over the storage limit, the idea of restarting may be adopted. One can run JD for a finite number of (say k) steps, then retain ℓ ($\ell < k$) vectors from the constructed subspace, and restart the JD iteration by augmenting the remaining subspace (spanned by the retained vectors) with new correction vectors.

6.1.5 Generalized Eigenvalue Problems

The basic formulation introduced above applies to a generalized eigenvalue problem

$$Kx = \lambda Mx$$

also. The analog to (6.2) is

$$K(u + z) - (\theta + \gamma)M(u + z) = 0 \quad \text{and} \quad u^H z = 0.$$

After dropping the second order term γMu, one has

$$(K - \theta M)z - \gamma Mu = -r, \quad \text{where} \quad r = Ku - \theta Mu.$$

If we replace the orthogonal scaling $u^H z = 0$ with $\tilde{u}^H z = 0$ for some $\tilde{u} \notin u^\perp$, the correction equation can be expressed in the following bordered form:

$$\begin{pmatrix} K - \theta M & Mu \\ \tilde{u}^H & 0 \end{pmatrix} \begin{pmatrix} z \\ -\gamma \end{pmatrix} = \begin{pmatrix} -r \\ 0 \end{pmatrix}. \tag{6.11}$$

If θ is computed by imposing the Petrov-Galerkin condition:

$$w^H(K - \theta M)u = 0,$$

one can also rewrite (6.11) as

$$(I - \frac{\tilde{w}w^H}{w^H\tilde{w}})(K - \theta M)(I - \frac{u\tilde{u}^H}{\tilde{u}^H u})z = -r,$$

where $\tilde{w} = Mu$. From this point, a JD algorithm for a generalized eigenvalue problem can be developed in the same way a standard JD algorithm is concocted.

6.2 JDQR and JDQZ

The earlier version (Figure 6.1) of the JD algorithm is presented merely to illustrate the most important aspects of the method. In this section, we modify that algorithm to make it suitable for computing several eigenpairs. To promote numerical stability, we seek a *partial Schur form*

$$AQ_k = Q_k R_k,$$

where $Q_k \in \mathbb{C}^{n\times k}$ satisfies $Q_k^H Q_k = I_k$ and $R_k \in \mathbb{C}^{k\times k}$ is upper triangular. The columns of Q_k are often called Schur vectors. They form an orthonormal basis for a k-dimensional invariant subspace of A. The diagonal of R_k consists of k eigenvalues of A.

Suppose we have computed

$$AQ_{k-1} = Q_{k-1}R_{k-1}.$$

Our next step is to extend the invariant subspace from Q_{k-1} to Q_k by finding the next Schur vector q_k that will bring in the desired spectral information.

Given an initial guess $\tilde{q}_k$, the algorithm to be described below generates a sequence of vectors z_i $(i = 1, 2, ...)$ which, together with $\tilde{q}_k$, form a subspace $\mathcal{S}$ from which a better approximation to q_k can be extracted. We will first demonstrate how one particular z_i is generated. For simplicity, we omit the subscript of z_i in the following discussion. Each z is generated to correct the approximate Schur vector $\tilde{q}_k$. In fact, we need a correction pair (z, γ) that satisfies

$$A(Q_{k-1}, \tilde{q}_k + z) = (Q_{k-1}, \tilde{q}_k + z) \begin{pmatrix} R_{k-1} & \hat{h} \\ 0 & \theta + \gamma \end{pmatrix}, \tag{6.12}$$

where $\theta = \tilde{q}_k^H A \tilde{q}_k$ and $\hat{h} = Q_{k-1}^H A(\tilde{q}_k + z)$.

The k-th column of (6.12) reads

$$A(\tilde{q}_k + z) = Q_{k-1}\hat{h} + (\tilde{q}_k + z)(\theta + \gamma).$$

As before, we drop the second order term γz to arrive at a correction equation

$$(A - \theta I)z - Q_{k-1}\hat{h} - \tilde{q}_k\gamma = -(A - \theta I)\tilde{q}_k.$$

Let's put $r = (A - \theta I)\tilde{q}_k$, $\tilde{Q}_k = (Q_{k-1}, \tilde{q}_k)$ and $g = (\hat{h}, \gamma)^T$. With the orthogonality constraint $\tilde{Q}_k^H z = 0$, we can now express the correction equation succinctly by

$$\begin{pmatrix} A - \theta I & \tilde{Q}_k \\ \tilde{Q}_k^H & 0 \end{pmatrix} \begin{pmatrix} z \\ -g \end{pmatrix} = \begin{pmatrix} -r \\ 0 \end{pmatrix}, \tag{6.13}$$

or equivalently by

$$(I - \tilde{Q}_k\tilde{Q}_k^H)(A - \theta I)(I - \tilde{Q}_k\tilde{Q}_k^H)z = -r. \tag{6.14}$$

As in the earlier version of the JD algorithm, one can solve the correction equation by either a direct or an iterative method to complete the "Jacobi" phase of the algorithm.

The solution z is added into a *search space* $\mathcal{S}$ which initially consists of only $\tilde{q}_k$. We follow the Rayleigh-Ritz procedure and approximate q_k from $\mathcal{S}$ by first computing a Schur decomposition of $G = V^H A V$, where $V = (\tilde{q}_k, z)$. Suppose y is the Schur vector of G corresponding to the desired Ritz value θ. We then form

$$\tilde{q}_k^+ = Vy,$$

to fulfill the "Davidson" part of the job. If the residual

$$r = (I - Q_{k-1}Q_{k-1}^H)(A - \theta I)\tilde{q}_k^+$$

is sufficiently small, we declare $\tilde{q}_k^+$ as converged. Otherwise, we set

$$\tilde{q}_k \leftarrow \tilde{q}_k^+, \quad \tilde{Q}_k \leftarrow (Q_{k-1}, \tilde{q}_k),$$

and return to equation (6.13) for a new correction. The correction vector will be used to expand $\mathcal{S}$ from which further improvements to $\tilde{q}_k$ can be made. To maintain numerical stability, we orthogonalize z against all columns of V and normalize it to have unit length before setting $V \leftarrow (V, z)$.

This JD process is continued until all desired eigenvalues have been found or when the dimension of $\mathcal{S}$ has reached the storage limit. In the latter case, we may discard part of V, retain j_{min} basis vectors and restart the JDQR process. The details are explained in Step 3.3 of the algorithm given in Figure 6.2. In that description, we use the MATLAB notation $U(:, 1 : j_{min})$ to denote the first j_{min} columns of U.

Since the Schur decomposition for the projected problem

$$G = V^H A V$$

is computed by the conventional QR algorithm, the modified JD algorithm is referred to as JDQR. We outline the basic structure of the algorithm in Figure 6.2 and refer the interested reader to [21] for implementation details.

Algorithm: JDQR

Input: a matrix A, a starting vector v_0
Output: A partial Schur form $AQ_k = Q_k R_k$, where $Q_k^H Q_k = I_k$, and
R_k is upper triangular. The diagonal of R_k consists eigenvalues of A.

1. $v \leftarrow v_0/\|v_0\|$, $\theta \leftarrow v^H A v$;
2. $V \leftarrow (v)$; $H \leftarrow (\theta)$; $Q \leftarrow ()$; $\tilde{Q} \leftarrow (v)$; $j \leftarrow 1$;
3. while (not all converged) **do**
  3.1 while $j < j_{max}$
    3.1.1 Solve (approximately)

$$\begin{pmatrix} A - \theta I & \tilde{Q} \\ \tilde{Q}^H & 0 \end{pmatrix} \begin{pmatrix} z \\ -g \end{pmatrix} = \begin{pmatrix} -r \\ 0 \end{pmatrix};$$

    3.1.2 $z \leftarrow (I - VV^H)z$;
    3.1.3 $V \leftarrow (V, z)$; $H \leftarrow V^H A V$;
    3.1.4 Compute the Schur decomposition of $H = USU^H$;
    3.1.5 Choose the desired Schur vector s, and Ritz value θ;
    3.1.6 Put $\tilde{q} \leftarrow Vs$;
    3.1.7 If ($\tilde{q}$ has converged to an Schur vector of A) **then**
      3.1.7.1 Set $Q \leftarrow (Q, \ \tilde{q})$;
      3.1.7.2 $V \leftarrow VU(:, 2:j)$;
    3.1.8 else
      3.1.8.1 Set $\tilde{Q} \leftarrow (Q, \ \tilde{q})$;
    3.1.9 endif
    3.1.10 $j \leftarrow j + 1$;
  3.2 end while;
  3.3 if (not all converged) **then**
    3.3.1 $V = VU(:, 1:j_{min})$;
    3.3.2 $j \leftarrow j_{min}$;
  3.4 endif
4. end while;

Figure 6.2 The Jacobi-Davidson QR Iteration

For a generalized eigenvalue problem

$$Kx = \lambda Mx,$$

it is desirable to compute a *generalized Schur form*:

$$KQ_k = W_k S_k, \quad MQ_k = W_k T_k, \tag{6.15}$$

where $Q_k, W_k \in \mathbb{C}^{n\times k}$. Matrices Q_k and W_k satisfy

$$Q_k^H Q_k = I_k \ \text{ and } \ W_k^H W_k = I_k.$$

Columns of Q_k and W_k are often referred to as the *right* and *left* Schur vectors respectively. Both $S_k \in \mathbb{C}^{k\times k}$ and $T_k \in \mathbb{C}^{k\times k}$ are upper triangular. The diagonals of S_k and T_k form k *generalized eigenvalue* pairs (α_j, β_j) $(j = 1, 2, ..., k.)$ Notice that if $\alpha_j \neq 0$ then $\lambda_j = \beta_j/\alpha_j$.

In the following, we will illustrate the steps required to extend the generalize partial Schur form from

$$KQ_{k-1} = W_{k-1} S_{k-1}, \quad MQ_{k-1} = W_{k-1} T_{k-1}$$

to (6.15). We will focus on computing the right Schur vector q_k such that $Q_k = (Q_{k-1}, q_k)$. Once we have q_k, the corresponding left Schur vector can be easily calculated by

$$\begin{aligned} w &\leftarrow (I - W_{k-1}W_{k-1}^H)Kq_k \ \text{ and } \ w_k \leftarrow \frac{w}{\|w\|}, \\ \text{or } \ w &\leftarrow (I - W_{k-1}W_{k-1}^H)Mq_k \ \text{ and } \ w_k \leftarrow \frac{w}{\|w\|}. \end{aligned}$$

The strategy given below for computing q_k is in the same spirit as the correction scheme used in JDQR. Since the generalized eigenpair (α, β) associated with the right

Schur vector q_k must satisfy

$$Kq_k = W_{k-1}s + w_k\alpha \quad \text{and} \quad Mq_k = W_{k-1}t + w_k\beta,$$

for some s and t, it is easy to deduce from above that

$$\alpha = \|(I - W_{k-1}W_{k-1}^H)Kq_k\|, \quad \beta = \|(I - W_{k-1}W_{k-1}^H)Mq_k\|.$$

We assume an approximate Schur vector $\tilde{q}_k$ is available. An approximation to (α, β) can be formed by

$$\tilde{\alpha} = \|(I - W_{k-1}W_{k-1}^H)K\tilde{q}_k\|, \quad \tilde{\beta} = \|(I - W_{k-1}W_{k-1}^H)M\tilde{q}_k\|.$$

We are interested in finding the correction triplet $(z, \Delta\alpha, \Delta\beta)$ such that

$$K(Q_{k-1}, \tilde{q}_k + z) = (W_{k-1}, w_k) \begin{pmatrix} S_{k-1} & \hat{s} \\ 0 & \tilde{\alpha} + \Delta\alpha \end{pmatrix}, \tag{6.16}$$

$$M(Q_{k-1}, \tilde{q}_k + z) = (W_{k-1}, w_k) \begin{pmatrix} T_{k-1} & \hat{t} \\ 0 & \tilde{\beta} + \Delta\beta \end{pmatrix}, \tag{6.17}$$

where $\hat{s} = W_{k-1}^H K(\tilde{q}_k + z)$, $\hat{t} = W_{k-1}^H M(\tilde{q}_k + z)$ and w_k is a left Schur vector that can be easily determined once we know z. Too many unknowns appear in the last columns of the above equations:

$$K(\tilde{q}_k + z) = W_{k-1}\hat{s} + w_k(\alpha + \Delta\alpha) \tag{6.18}$$

$$M(\tilde{q}_k + z) = W_{k-1}\hat{t} + w_k(\beta + \Delta\beta) \tag{6.19}$$

To eliminate the term involving w_k, we multiply (6.18) by $\beta + \Delta\beta$ and (6.19) by $\alpha + \Delta\alpha$ to obtain

$$(\beta + \Delta\beta)K(\tilde{q}_k + z) - (\alpha + \Delta\alpha)M(\tilde{q}_k + z) = W_{k-1}\Big[\hat{s}(\beta + \Delta\beta) - \hat{t}(\alpha + \Delta\alpha)\Big]. \tag{6.20}$$

It follows from (6.16) and (6.17) that $W_{k-1} \in \text{span}\{KQ_{k-1}, MQ_{k-1}\}$. Thus, if we put $\tilde{Q}_k = (Q_{k-1}, \tilde{q}_k)$, it is easy to verify that

$$\Delta\beta K\tilde{q}_k - \Delta\alpha M\tilde{q}_k - W_{k-1}\left[\hat{s}(\beta+\Delta\beta) - \hat{t}(\alpha+\Delta\alpha)\right] = (\mu K + \nu M)\tilde{Q}_k g,$$

for some μ, ν and $g \in \mathbb{C}^{k\times 1}$. (We will define μ and ν shortly.) Based on this simplification, we can now rewrite (6.20), after dropping the second order terms $\Delta\alpha Kz$ and $\Delta\beta Mz$, as

$$(\tilde{\alpha}K - \tilde{\beta}M)z - (\mu K + \nu M)\tilde{Q}_k g = -r,$$

where $r = (\tilde{\alpha}K - \tilde{\beta}M)\tilde{q}_k$. Together with the orthogonality constraint $\tilde{Q}_k^H z = 0$, we can formulate the correction equation as

$$\begin{pmatrix} \tilde{\alpha}K - \tilde{\beta}M & Y_k \\ \tilde{Q}_k & 0 \end{pmatrix} \begin{pmatrix} z \\ -g \end{pmatrix} = \begin{pmatrix} -r \\ 0 \end{pmatrix}, \tag{6.21}$$

where $Y_k = (\mu K + \nu M)\tilde{Q}_k$. An equivalent representation is the projected equation:

$$(I - Y_k Y_k^H)(\tilde{\alpha}K - \tilde{\beta}M)(I - \tilde{Q}_k\tilde{Q}_k^H)z = -r.$$

We have purposely neglected to comment on the choice of μ and ν to avoid cluttering the derivation of the correction equation. The exact values of these parameters are not important as long as the approximation to the left Schur vector w_k is well represented in Y_k; as $\tilde{Q}_k$ converges to Q_k, so should $\text{span}\{Y_k\}$ approach to $\text{span}\{W_k\}$. It is suggested in [21] that one should choose μ and ν so that

$$\varrho = \left\|(I - W_{k-1}W_{k-1}^H)(\mu K\tilde{q}_k + \nu M\tilde{q}_k)\right\|$$

is maximized. If $\tilde{q}_k$ is near q_k, it is reasonable to use

$$\mu = \frac{\bar{\tilde{\alpha}}}{\sqrt{|\tilde{\alpha}|^2 + |\tilde{\beta}|^2}} \quad \text{and} \quad \nu = \frac{\bar{\tilde{\beta}}}{\sqrt{|\tilde{\alpha}|^2 + |\tilde{\beta}|^2}}, \tag{6.22}$$

because in this case

$$\varrho \approx \left\| (I - W_{k-1}W_{k-1}^H)(\mu K q_k + \nu M q_k) \right\| = |\mu\alpha + \nu\beta|$$

will be maximal if μ and ν are defined as in (6.22). If $\tilde{q}_k$ is not accurate enough, one could use

$$\mu = \frac{\bar{\sigma}}{\sqrt{1+|\sigma|^2}} \quad \text{and} \quad \nu = \frac{1}{\sqrt{1+|\sigma|^2}},$$

where σ is a target shift.

Once the correction z is computed, one can use it to augment the search space $V = \text{span}\{\tilde{q}_k, z\}$ from which the next approximation to q_k is drawn. In the meantime, one obtains a correction to the approximate left Schur vector

$$\tilde{w}_k = \frac{(I - W_{k-1}W_{k-1}^H)(\mu K \tilde{q}_k + \nu M \tilde{q}_k)}{\|(I - W_{k-1}W_{k-1}^H)(\mu K \tilde{q}_k + \nu M \tilde{q}_k)\|}$$

by

$$f \leftarrow (I - \tilde{w}_k \tilde{w}_k^H)(\mu K z + \nu M z).$$

Now put $L = (\tilde{w}_k, f)$. New Schur approximations can be computed by invoking the Petrov-Galerkin condition:

$$L^H(\tilde{\alpha} K - \tilde{\beta} M) V y = 0.$$

We omit the implementation details here and refer the interested reader to [21] for a complete JDQZ algorithm.

6.3 Solving the Correction Equation

We hinted earlier that the convergence rate of JDQR and JDQZ depends, to a large extent, on how efficiently and accurately the correction equation can be solved. To

keep our discussion uncomplicated, we will focus only on issues related to solving the correction equation

$$\begin{pmatrix} A-\theta I & \tilde{Q}_k \\ \tilde{Q}_k^H & 0 \end{pmatrix} \begin{pmatrix} z \\ -g \end{pmatrix} = \begin{pmatrix} -r \\ 0 \end{pmatrix} \tag{6.23}$$

that appears in a JDQR iteration.

If one can afford to factor $A-\theta I$, direct methods based on Gauss elimination are usually the best choice for solving (6.23). It follows from the block factorization

$$\begin{pmatrix} A-\theta I & \tilde{Q}_k \\ \tilde{Q}_k^H & 0 \end{pmatrix} = \begin{pmatrix} I & 0 \\ \tilde{Q}_k^H(A-\theta I)^{-1} & I \end{pmatrix} \begin{pmatrix} A-\theta I & \tilde{Q}_k \\ 0 & -\tilde{Q}_k^H(A-\theta I)^{-1}\tilde{Q}_k \end{pmatrix}$$

that

$$z = (A-\theta I)^{-1}r - (A-\theta I)^{-1}\tilde{Q}_k G_k^{-1} y_k,$$

where $G_k = \tilde{Q}_k^H(A-\theta I)^{-1}\tilde{Q}_k$ and $y_k = \tilde{Q}_k^H(A-\theta I)^{-1}r$. This implies that $k+1$ linear systems with the same coefficient matrix $A-\theta I$ must be solved to produce a correction vector z. Notice that this is in sharp contrast to the solution of the TRQ equation which requires solving only one linear equation associated with $A-\theta I$.

Solving (6.13) using direct methods usually results in rapid convergence of the approximating Schur vectors. In terms of number of iterations, it is typical to see quadratic convergence for non-symmetric problems and cubic convergence for symmetric problems. However, the original intent for developing JD type of algorithm is to allow preconditioned iterative solvers to be used to produce inexact Newton directions.

Iterative solvers of the Krylov type (such as GMRES[76], QMR[24], BiCG-STAB[16], SYMMLQ[55] etc.) can be easily applied to the alternative representation of the correction equation

$$(I-\tilde{Q}_k\tilde{Q}_k^H)(A-\theta I)(I-\tilde{Q}_k\tilde{Q}_k^H)z = -r.$$

The difficulty is to construct a reasonable preconditioner for this projected equation. By a preconditioner, we mean a matrix that approximates the coefficient of the linear system in some sense. If the eigenvalue problem originates from a discretization of some differential operator, one often has a good preconditioner for the matrix A or $A - \theta I$, say P. We would like to take advantage of this preconditioner to construct an approximate inverse for the projected matrix

$$B = (I - \tilde{Q}_k \tilde{Q}_k^H)(A - \theta I)(I - \tilde{Q}_k \tilde{Q}_k^H).$$

It is suggested in [21] that a reasonable preconditioner takes the form

$$\hat{P} = (I - \tilde{Q}_k \tilde{Q}_k^H) P (I - \tilde{Q}_k \tilde{Q}_k^H). \tag{6.24}$$

The following lemma gives two explicit formulae for an inverse of $\hat{P}$. (Note that the inverse of $\hat{P}$ is not unique since $\hat{P}$ is singular.)

Lemma 6.1 An inverse of $\hat{P} = (I - \tilde{Q}_k \tilde{Q}_k^H) P (I - \tilde{Q}_k \tilde{Q}_k^H)$ can be expressed as

$$\begin{aligned} \hat{P}^{-1} &= \left[I - \tilde{Y}_k G_k^{-1} \tilde{Q}_k^H \right] P^{-1}, \\ \text{or} \quad \hat{P}^{-1} &= P^{-1} \left[I - \tilde{Q}_k G_k^{-1} \tilde{U}_k^H \right], \end{aligned}$$

where $G_k = \tilde{Q}_k^H P^{-1} \tilde{Q}_k$, $\tilde{Y}_k = P^{-1} \tilde{Q}_k$ and $\tilde{U}_k = P^{-H} \tilde{Q}_k$.

Proof To reveal the structure of the $\hat{P}^{-1}$, we examine how the equation

$$\hat{P} x = b, \quad \text{where} \quad x, b \in \tilde{Q}_k^{\perp} \tag{6.25}$$

can be solved. After rewriting (6.25) in its border form

$$\begin{pmatrix} P & \tilde{Q}_k \\ \tilde{Q}_k^H & 0 \end{pmatrix} \begin{pmatrix} x \\ g \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix},$$

we apply block LU factorization to get

$$\begin{pmatrix} I & 0 \\ \tilde{Q}_k^H P^{-1} & I \end{pmatrix} \begin{pmatrix} P & \tilde{Q}_k \\ 0 & -\tilde{Q}_k^H P^{-1} \tilde{Q}_k \end{pmatrix} \begin{pmatrix} x \\ g \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix}.$$

It is easy to deduce, after back substitution, that

$$\begin{aligned} g &= \left(\tilde{Q}_k^H P^{-1} \tilde{Q}_k\right)^{-1} \tilde{Q}_k^H P^{-1} b = G_k^{-1} \tilde{Q}_k^H P^{-1} b, \\ x &= P^{-1}\left(b - \tilde{Q}_k G_k^{-1} \tilde{Q}_k^H P^{-1} b\right) \\ &= P^{-1}\left(I - \tilde{Q}_k G_k^{-1} \tilde{Q}_k^H P^{-1}\right) b \qquad (6.26) \\ &= \left(I - P^{-1} \tilde{Q}_k G_k^{-1} \tilde{Q}_k^H\right) P^{-1} b \qquad (6.27) \end{aligned}$$

Using the definition $\tilde{U}_k = P^{-H} \tilde{Q}_k$ and $\tilde{Y}_k = P^{-1} \tilde{Q}_k$, we conclude from (6.26) and (6.27) that

$$\begin{aligned} \hat{P}^{-1} &= \left[I - \tilde{Y}_k G_k^{-1} \tilde{Q}_k^H\right] P^{-1}, \\ \text{or } \hat{P}^{-1} &= P^{-1}\left[I - \tilde{Q}_k G_k^{-1} \tilde{U}_k^H\right]. \end{aligned}$$

□

With this preconditioner, we can now apply a Krylov subspace solver to the preconditioned equation

$$\hat{P}^{-1} B z = -\hat{P}^{-1} r.$$

Notice that

$$\hat{P}^{-1} B = P^{-1}\left[I - \tilde{Q}_k G_k^{-1} \tilde{U}_k^H\right](I - \tilde{Q}_k \tilde{Q}_k^H)(A - \theta I)(I - \tilde{Q}_k \tilde{Q}_k^H). \qquad (6.28)$$

Using the fact that

$$\left(I - \tilde{Q}_k G_k^{-1} \tilde{U}_k^H\right)\left(I - \tilde{Q}_k \tilde{Q}_k^H\right) = I - \tilde{Q}_k G_k^{-1} \tilde{U}_k^H,$$

we can further simplify (6.28) to obtain

$$\begin{aligned}\hat{P}^{-1}B &= P^{-1}\left[(I-\tilde{Q}_k G_k^{-1}\tilde{U}_k^H\right](A-\theta I)(I-\tilde{Q}_k\tilde{Q}_k^H)\\ &= \hat{P}^{-1}(A-\theta I)(I-\tilde{Q}_k\tilde{Q}_k^H)\\ &= \left[I-\tilde{Y}_k G_k^{-1}\tilde{Q}_k^H\right]P^{-1}(A-\theta I)(I-\tilde{Q}_k\tilde{Q}_k^H).\end{aligned}$$

Since we seek a correction vector $z \in \tilde{Q}_k^{\perp}$,

$$z = \left(I-\tilde{Q}_k\tilde{Q}_k^H\right)z = \left(I-\tilde{Y}_k G_k^{-1}\tilde{Q}_k^H\right)z.$$

Thus, we can also express the preconditioned correction equation as

$$\left(I-\tilde{Y}_k G_k^{-1}\tilde{Q}_k^H\right)P^{-1}(A-\theta I)\left(I-\tilde{Y}_k G_k^{-1}\tilde{Q}_k^H\right)z = -\hat{r},$$

where

$$\hat{r} = \hat{P}^{-1}r = \left(I-\tilde{Y}_k G_k^{-1}\tilde{Q}_k^H\right)r.$$

A Krylov solver applied to this equation generates an orthonormal basis

$$\{v_1, v_2, ..., v_k\}$$

for $\mathcal{K}(\hat{P}^{-1}B, v_1; k)$. To calculate v_j, one need not form $(I-\tilde{Y}_j G_j^{-1}\tilde{Q}_j^H)v_{j-1}$ since $\tilde{Q}_j^H v_j = 0$ has already been enforced by the previous calculation. More implementation details can be found in [21].

We shall mention that the TRQ equation (3.6) developed in Chapter 3 has the same pattern as correction equation (6.13). However, since the matrix V_k in the TRQ equation satisfies an Arnoldi relation

$$AV_k = V_k H_k + f_k e_k^T,$$

the TRQ equation can be further simplified to reduce the amount of work for solving (3.6) to essentially that of solving a single equation of the form $(A-\theta I)w = v$. (See

Lemma 3.4.) This simplification is not possible for the correction equation (6.13). As to an iterative solver, one can use a preconditioner developed for $A - \theta I$ directly in TRQ without further projections.

6.4 Comparison with ITRQ

In this section, we compare JDQR with ITRQ through a numerical example. Both methods are applied to the matrix CK656 [6] to extract four eigenvalues nearest the target shift $\sigma = 5.0$.

In ITRQ, the dimension of the Krylov subspace is set to $k = 5$. In JDQR, the maximum number of basis vectors generated (that is, the maximum number of columns of V in Figure 6.2) is $j_{max} = 8$. If convergence does not occur after all j_{max} basis vectors have been generated, the first $j_{min} = 4$ columns of V are retained and JDQR restarts from the fifth column of V. Both methods used unpreconditioned GMRES to solve the linear system. The maximum number of GMRES steps allowed is set to 10. The GMRES convergence tolerance is set to 10^{-6}. Convergence a Ritz pair is declared when its residual norm falls below $tol = 10^{-9}$.

We plot the convergence history of each Ritz pair in Figure 6.3. The residual norm of each Ritz pair is plotted against the number of flops used to obtain the pair. The residual norms are monitored one at a time, that is, the residual curve of the $j+1$st Ritz pair is shown only after the j-th Ritz pair has converged.

We observe that the overall performance of both method are almost the same. However, the inexact TRQ was able to capture the first eigenpair much quicker than JDQR. But Figure 6.3 also shows that after the initial construction of a proper subspace, JDQR convergence rapidly. In particular, the second and third eigenvalue converge at almost the same time.

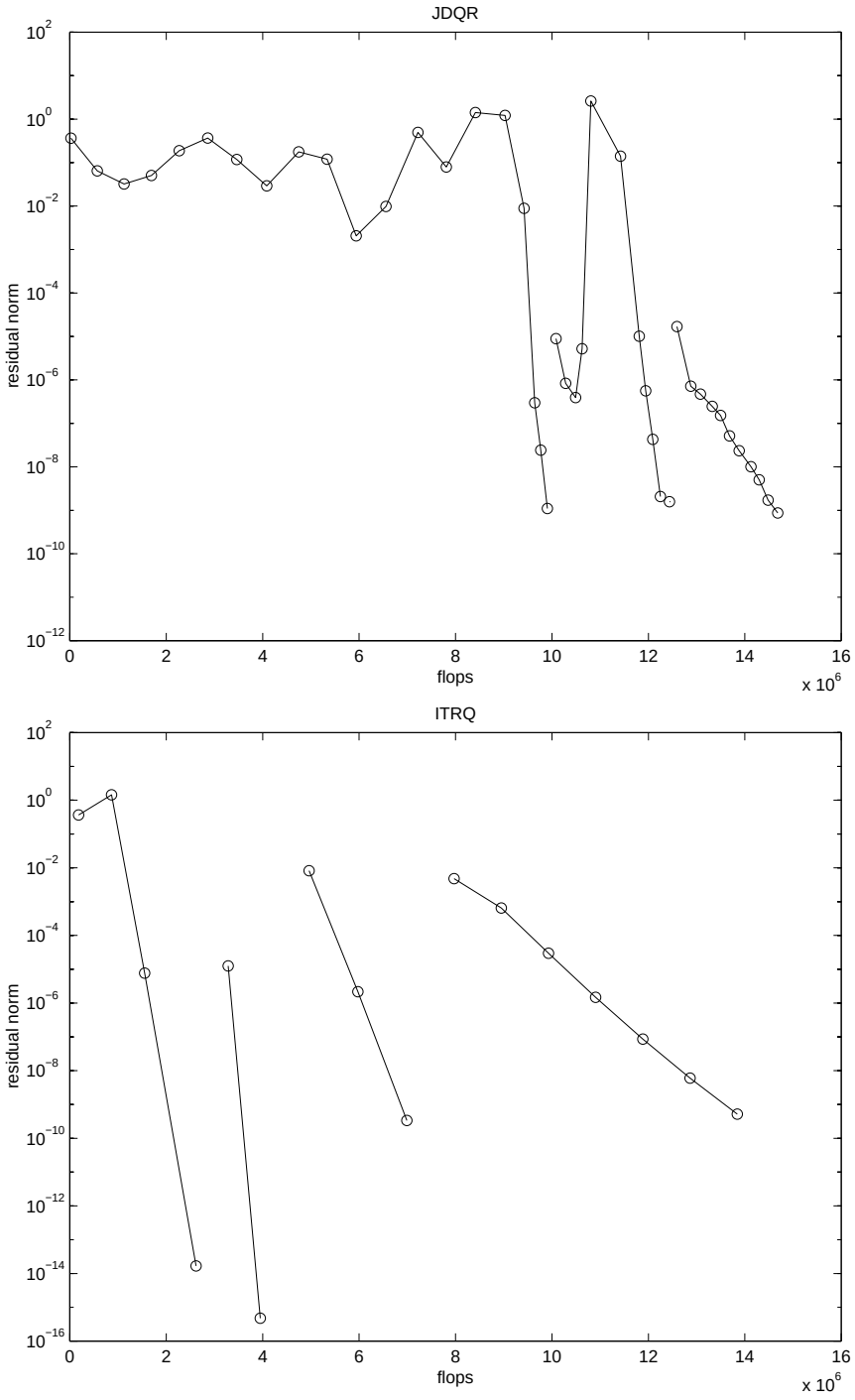

Figure 6.3 The convergence history of JDQR and ITRQ for the CK656 matrix.

Chapter 7

Thesis Summary and Future Work

The main purpose of this thesis is to investigate various ways of accelerating the Arnoldi iteration, a method widely used to compute eigenvalues and eigenvectors of a large sparse and/or structured matrix. The acceleration strategies discussed fall roughly into the following three groups:

- the method of restarting;
- the method of spectral transformation;
- the method of eigenvector or Schur vector correction;

The first two strategies are applied within the Arnoldi iteration itself. They are devised to enhance either the starting vector or the distribution of the eigenvalues, two major factors that determine the convergence of the Arnoldi process. The third strategy accelerates the eigenvalue calculation using a different approach. The iteration is not carried out within a Krylov subspace. Instead it takes an eigenvector or Schur vector approximation obtained from an Arnoldi iteration and further improves it by adding orthogonal corrections.

The first two methods both have a polynomial and a rational version. The advantage of the polynomial version is apparent. It only requires matrix vector multiplications during the accelerated Arnoldi iteration, whereas a rational acceleration must, in one way or another, accurately solve a sequence of linear systems accurately. The method of restarting is elegant in the sense that it can be viewed as a truncated version of the full QR or RQ-iteration about which we have a good understanding

both in theory and in practice. The method of spectral transformation is conceptually simple, yet it is often the method of choice when a large number of interior eigenvalues are requested. It must be implemented with care to be cost-effective.

The method of rational restarting (via TRQ) and the method of Newton correction (via JDQR) are derived from two rather different contexts. However, they share many similar properties and difficulties. In particular, both method must solve a bordered system

$$\begin{pmatrix} A-\theta I & X_k \\ X_k^H & 0 \end{pmatrix} \begin{pmatrix} z \\ g \end{pmatrix} = \begin{pmatrix} r \\ 0 \end{pmatrix}. \tag{7.1}$$

In TRQ, $X_k \in \mathbb{C}^{n\times k}$ consists of the Arnoldi vectors. This allows the bordered equation to be simplified. The simplification reduces the amount of work to essentially that of solving a single linear system of the form $(A-\theta I)w = v$. Since the Arnoldi structure is absent in JDQR, it is not possible to make such a simplification. If a direct linear solver is used, the amount of work required is equivalent to solving k linear systems with the same coefficient $A-\theta I$.

It is possible, in both TRQ and JDQR, to solve (7.1) iteratively. In fact, one of the motivations for developing these methods is to incorporate a preconditioned iterative solver into the eigenvalue calculation so that rapid convergence can be achieved without factoring matrices. In TRQ, one can apply the iterative solver directly to the simplified equation $(A-\theta I)w = v$, whereas in JDQR, it is preferable to apply an iterative solver to the alternative form of (7.1):

$$(I - X_k X_k^H)(A-\theta I)(I - X_k X_k^H)z = r. \tag{7.2}$$

The main advantage of this approach is that the approximate solution is produced within the subspace $X_k^{\perp}$.

The construction of a good preconditioner is difficult in both algorithms, especially when θ is near an eigenvalue of A. In Chapter 6, we have demonstrated how to

construct a preconditioner for the projected system (7.2) given a good preconditioner of $A - \theta I$.

In TRQ, the error produced during the process of solving the TRQ equation propagates into the Arnoldi basis by the rotations used to complete an implicit RQ update. As a consequence, one must rebuild the Arnoldi factorization from the very first column each time a TRQ equation is solved. However, since the error is also damped by those rotations, the first Arnoldi basis vector still converges to an eigenvector of A under some appropriate assumptions. It has been shown in Chapter 3 that the convergence is locally linear. In JDQR, the relaxation of the solution accuracy results in an inexact Newton scheme for correcting the Schur vector approximation. The convergence of the eigenvalue calculation can be explained by the standard arguments for the inexact Newton method.

The JDQR algorithm can be generalized to the JDQZ algorithm for solving a generalized eigenvalue problem. Unfortunately, it is difficult to make such a generalization for TRQ. Recently, Sorensen has developed some iterative schemes for solving generalized eigenvalue problems based on a truncated QZ iteration [82]. Just like JDQZ, these algorithms do not belong to the family of Krylov subspace methods.

The numerical examples presented in this thesis have indicated that the difficulties in large scale eigenvalue calculation can be characterized as the following:

1. In the symmetric case, computing interior eigenvalues without factoring matrices poses a major challenge. Inexact TRQ, JDQR and IRL with polynomial spectral transformation are all possible candidates to use, but none of these has an apparent advantage over the others. For inexact TRQ and JDQR, constructing an effective preconditioner is the most crucial part of the algorithm. Good preconditioners are likely to be problem dependent. Future research in this direction requires a thorough understanding of the application from which

the eigenvalue problem arises in addition to the algebraic properties of the linear system. As to polynomial spectral transformation, heuristics are yet to be developed to help determine the type of polynomial to use and the optimal degree that one should choose.

2. Non-symmetric eigenvalue problems share the same difficulty with the symmetric problems. In addition, the problem of computing the rightmost eigenvalues requires special attention. Even when one can afford to factor the matrix $A-\sigma I$ or $K-\sigma M$, computing the *spectral abscissa* (eigenvalue with the largest real part) can be rather difficult. This is because the desired eigenvalue can lie in any part of the complex plane, making it difficult for one to choose a target shift in advance. One particular example in which this difficulty arises is the work of Cox and Zuazua [12]. They tried to identify the minimum rate at which energy decays in a damped string. This rate can be characterized as the spectral abscissa of the operator

$$\mathcal{A} = \begin{pmatrix} 0 & I \\ d^2/dx^2 & -2a(x) \end{pmatrix},$$

where $a(x)$ represents the damping coefficient. The spectrum of the discretized operator corresponding to the damping $a(x) = 2 + \cos(x)$ is shown in Figure 7.1. (we omit the discretization details here to illustrate the main point of the problem.) The real parts of all eigenvalues lie between -3 and -2.75 while the imaginary parts vary from -210 to 210. It is difficult to use IRA to separate the rightmost eigenvalue from the rest of the spectrum because eigenvalues are clustered in the real axis direction. Unless one knows approximately where the spectral abscissa is a priori, it is difficult to guess where to place a target shift for a shift-invert Arnoldi iteration.

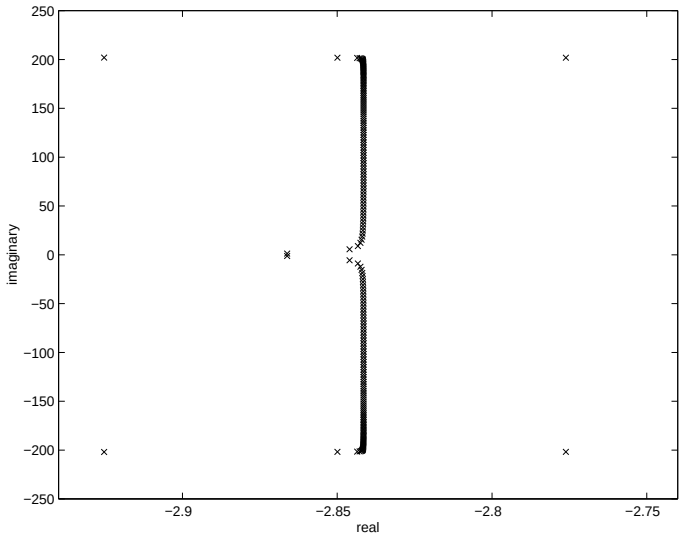

Figure 7.1 The spectrum of a discretized damped string operator

The acceleration strategies considered in this thesis are purely algebraic. That is, we have ignored the nature of the original problem and focused only on the algebraic properties of the matrices we are handed. Since many eigenvalue problems arise from the discretization of linear differential or integral operators, one can benefit a great deal from a thorough understanding of the properties of these operators in developing acceleration schemes for the Arnoldi iteration. In particular, it is possible to develop some multilevel acceleration schemes which first solve the continuous model on a coarse grid, then interpolate the approximate eigenvectors on a finer grid to form a starting guess for the next level Arnoldi iteration. Methods of this type have been investigated in [28, 44, 7, 42, 45], but the approach taken there is based on a (single vector) inverse iteration in which the linear equation solved by a multi-grid method. One shall expect a better convergence rate from a multilevel Arnoldi iteration.

Bibliography

[1] P. W. Anderson. Absence of diffusion in certain random lattices. *Phys. Rev.*, 109:1492–1505, 1958.

[2] W. E. Arnoldi. The principle of minimized iterations in the solution of the matrix eigenvalue problem. *Quarterly of Applied Mathematics*, 9:17–29, 1951.

[3] S. F. Ashby. *Polynomial Preconditioning for Conjugate Gradient Methods.* PhD thesis, Dept. of Comp. Sci., University of Illinois-Urbana Champaign, Urbana, IL, 1987. Report No. UIUCDCS-R-87-1355.

[4] S. F. Ashby, T. A. Manteuffel, and J. S. Otto. A comparison of adaptive Chebyshev and least squares polynomial preconditioning for hermitian positive definite linear systems. *SIAM J. Sci. Stat. Comput.*, 13(1):1–29, 1992.

[5] K. E. Atkinson. *An Introduction to Numerical Analysis.* John Wiley and Sons, New York, 1978.

[6] Z. Bai, R. Barrett, D. Day, J. Demmel, and J. Dongarra. Test matrix collection (non-hermitian eigenvalue problems). Research report, Department of Mathematics, University of Kentucky, 1995.

[7] R. E. Bank. Analysis of a multlevel inverse iterative procedure for eigenvalue problems. *SIAM Journal on Numerical Analysis*, 19(5):886–897, 1982.

[8] K. Brattkus and D. I. Meiron. Numerical simulation of unsteady crystal growth. *SIAM Journal on Applied Mathematics*, 52(5):1303–1320, 1991.

[9] D. Calvetti, L. Reichel, and D. C. Sorensen. An implicitly restarted Lanczos method for large symmetric eigenvalue problems. *ETNA*, 2:1–21, March 1994.

[10] F. Chatelin. *Eigenvalues of Matrices*. Wiley, 1993.

[11] M. T. Chu. Note on the homotopy method for linear algebraic eigenvalue problems. *Linear Algebra and its Applications*, 105:225–236, 1988.

[12] S. J. Cox and E. Zuazua. The rate at which energy decays in a damped string. *Communications in Partial Differential Equations*, 19(1 and 2):213–243, 1994.

[13] J. Cullum and R. A. Wilboughby. *Lanczos algorithms for large symmetric eigenvalue computations*, volume 1 Theory. Birkhäuser, Boston, MA., 1985.

[14] J. Daniel, W. B. Gragg, L. Kaufman, and G. W. Stewart. Reorthogonalization and stable algorithms for updating the Gram–Schmidt QR factorization. *Mathematics of Computation*, 30:772–795, 1976.

[15] E. R. Davidson. The iterative calculation of a few of the lowest eigenvalues and corresponding eigenvectors of large real symmetric matrices. *Journal of Computational Physics*, 17:87–94, 1975.

[16] H. A. Van der Vorst. Bi-CGSTAB: A fast and smooth converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 13:631–644, 1992.

[17] I. D. Duff, R. G. Grimes, and J. G. Lewis. Sparse matrix test problems. *ACM Transactions on Mathematical Software*, 15:1–14, 1989.

[18] B. Fischer and R. W. Freund. On adaptive weighted polynomial precondtioning for hermitian positive definite matrices. *SIAM J. Sci. Comput.*, 15:408–426, 1994.

[19] B. Fischer and G. Golub. On generating polynomials which are orthogonal over several intervals. *Math. Comp.*, 56:711–730, 1991.

[20] B. Fisher and G. Golub. How to generate unknown orthogonal polynomials out of known orthogonal polynomials. *J. Comp. Appl. Math.*, 43:99–115, 1992.

[21] D. R. Fokkema, G. L. G. Sleijpen, and H.A. Van der Vorst. A Jacobi-Davidson style QR and QZ algorithm for partial reduction of matrix pencils. Technical Report 941, University Utrecht, Department of Mathematics, 1996.

[22] J. G. F. Francis. The QR transformation a unitary analogue to the LR transformation - part 1. *The Computer Journal*, 4:265–271, October 1961.

[23] J. G. F. Francis. The QR transformation - part 2. *The Computer Journal*, 4:332–345, January 1962.

[24] R. W. Freund and N. M. Nachtigal. QMR: a quasi-minimal residual method for non-Hermitian linear systems. *Numerische Mathematik*, 60:315–319, 1991.

[25] K. Gallivan and E. Grimme. A rational Lanczos algorithm for model reduction. *Numerical Algorithms*, 13:631–644, 1995.

[26] J. F. Grcar. *Analyses of the Lanczos Algorithm and of the Approximation Problem in Richardson's Method.* PhD thesis, Dept. of Comp. Sci., University of Illinois-Urbana Champaign, Urbana, IL, 1981.

[27] R. G. Grimes, J. G. Lewis, and H. D. Simon. A shifted block Lanczos algorithm for solving sparse symmetric generalized eigenproblems. *SIAM Journal on Matrix Analysis and Applications*, 15(1):228–272, January 1994.

[28] W. Hackbusch. On the computation of approximate eigenvalues and eigenvectors of elliptic operators by means of a multi-grid method. *SIAM Journal on Numerical Analysis*, 16:201–215, 1979.

[29] D. Ho. Tchebychev acceleration technique for large scale nonsymmetric matrices. *Numerische Mathematik*, 56:721–734, 1990.

[30] G. Horvay and J. W. Cahn. Dendritic and spheroidal growth. *Acta Netakkyrgica*, 9:695–705, July 1961.

[31] C. G. J. Jacobi. Ueber eine neue Auflösungsart derbei der Methode der kleinsten Quadrate vorkommende linearen Gleichungen. *Astronomische Nachrichten*, pages 297–306, 1845.

[32] O. G. Johnson, C. A. Micchelli, and G. Paul. Polynomial preconditioning for conjugate gradient calculation. *SIAM J. Numeri. Anal.*, 20:362–376, 1983.

[33] S. Kaniel. Estimates for some computational techniques in linear algebra. *Mathematics of Computation*, 20:369–378, 1966.

[34] A. Klein. Spreading of wave packets in the Anderson model on the Bethe lattice. *Communications in Mathematical Physics*, 177(3):755–773, 1996.

[35] M. N. Kooper, H. A. Van der Vorst, S. Poedts, and J. P. Goedbloed. Aplication of the implicitly updated arnoldi method with a complex shift and invert strategy in MHD. Technical report, FOM Rihinhuizen, Nieuwegin, The Netherlands, 1993. submitted to Journal of Computational Physics.

[36] J. S. Langer. Instabilities and pattern formation in crystal growth. *Reviews of Modern Physics*, 52(1):1–28, 1980.

[37] C. Lanzcos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *Journal of Research of the National Bureau of Standards*, 45(4):255–282, October 1950. Research Paper 2133.

[38] R. B. Lehoucq. *Analysis and Implementation of an Implicitly Restarted Iteration.* PhD thesis, Rice University, Houston, Texas, May 1995. Technical Report TR95-13, Dept. of Computational and Applied Mathematics.

[39] R. B. Lehoucq and K. Meerbergen. The inexact rational krylov sequence method. Technical report MCS-P612-1096, Argonne National Lab, Argonne, IL, 1996. To appear in SIAM Journal on Matrix Analysis and Applications.

[40] R. B. Lehoucq, D. C. Sorensen, P. Vu, and C. Yang. ARPACK: *An implementation of the Implicitly Re-started Arnoldi Iteration that computes some of the eigenvalues and eigenvectors of a large sparse matrix*, 1995. Available from ftp.caam.rice.edu under the directory pub/software/ARPACK.

[41] F. Leja. Sur certaines suites liées aux ensembles plan et leur application à la représentation conforme. *Ann. Polon. Math.*, 4:8–13, 1957.

[42] J. Mandel and S. McCormick. A multilevel variational method for $au = \lambda bu$ on composite grids. *Journal of Computational Physics*, 80:442–452, 1989.

[43] T. A. Manteuffel. Adaptive procedure for estimating parameters for the non-symmetric Tchebychev iteration. *Numerische Mathematik*, 31:183–208, 1978.

[44] S. F. McCormick. A mesh refinement method for $Ax = \lambda Bx$. *Mathematics of Computation*, 36(154):485–498, 1980.

[45] S. F. McCormick. Multilevel adaptive methods for elliptic eigenproblems: A two-level convergence theory. *SIAM Journal on Numerical Analysis*, 31(6):1731–1745, 1994.

[46] K. Meerbergen, A. Spence, and D. Roose. Shift-invert and Cayley transforms for the detection of rightmost eigenvalues of nonsymmetric matrices. *BIT*, 34:409–423, 1994.

[47] D. I. Meiron. Boundary integral formulation of the two-dimensional symmetric model of dendritic growth. *Physica D*, 23:329–339, 1986.

[48] F. Milde. Private communication, 1997.

[49] C. B. Moler and G. W. Stewart. An algorithm for generalized matrix eigenvalue problems. *SIAM Journal on Numerical Analysis*, 10(2):241–256, April 1973.

[50] R. B. Morgan. On restarting the Arnoldi method for large scale eigenvalue problems. *Mathematics of Computation*, 65(215):1213–1230, July 1996.

[51] W. W. Mullins and R. F. Sekerka. Morphological stability of a particle growing by diffusion or heat flow. *Journal of Applied Physics*, 34(2):323–329, 1963.

[52] W. W. Mullins and R. F. Sekerka. Stability of a planar interface during solidification of a dilute binary alloy. *Journal of Applied Physics*, 35(2):444–451, 1964.

[53] J. Olsen, P. Jorgensen, and J. Simons. Passing the one-billion limit in full configuration-interaction (fci) calculations. *Chemical Physics Letters*, 169:463–472, 1990.

[54] C. C. Paige. *The computation of eigenvalues and eigenvectors of very large sparse matrices*. PhD thesis, University of London, London, England, 1971.

[55] C. C. Paige and M. A. Saunders. Solutions of sparse indefinite systems of linear equations. *SIAM Journal on Numerical Analysis*, 12:617–629, 1975.

[56] T. W. Parks and C. S. Burrus. *Digital Filter Design.* John Wiley & Sons, 1987.

[57] B. N. Parlett. *The Symmetric Eigenvalue Problem.* Prentice-Hall, 1980.

[58] B. N. Parlett and D. Scott. The Lanczos algorithm with selective orthogonalization. *Mathematics of Computation*, 33:217–238, 1979.

[59] P. Pendergast, Z. Darakjian, E. F. Hayes, and D. C. Sorensen. Scalable algorithms for three-dimensional reactive scattering: Evaluation of a new algorithm for obtaining surface functions. *Journal of Computational Physics*, 113(2):201–214, 1994.

[60] G. Peters and J. H. Wilkinson. Inverse iteration, ill-conditioned equations and Newton's method. *SIAM Review*, pages 339–360, 1980.

[61] R. Radke. A MATLAB implementation of the implicitly restarted arnoldi method for solving large scale eigenvalue problems. Technical report, Dept. of Applied & Computational Mathematics, Rice University, Houston, TX, 1996. Master thesis.

[62] L. Reichel. The application of Leja points to Richardson iteration and polynomial preconditioning. *Linear Algebra Appl.*, 154-156:389–414, 1991.

[63] E. Y. Remez. Sur le calcul effectif des polynomies d'approximation de Tchebichef. *CR*, 199:337–340, 1934.

[64] T. D. Romo, J. B. Clarage, D. C. Sorensen, and Jr. G. N. Philips. Automatic identification of discrete substates in proteins: Singular value decomposition

analysis of time averagd crystallographic refinements. Technical Report CRPC-TR 94481, Center for Research on Parallel Computation, Rice University, Houston, TX, 1994.

[65] A. Ruhe. Algorithms for the nonlinear eigenvalue problem. *SIAM Journal on Numerical Analysis*, 4:674–689, 1973.

[66] A. Ruhe. Rational Krylov sequence methods for eigenvalue computations. *Linear Algebra and Its Applications*, 58:391–405, 1984.

[67] A. Ruhe. The rational Krylov algorithm for nonsymmetric eigenvalue problems, III: Complex shifts for real matrices. *BIT*, 34:165–176, 1994.

[68] A. Ruhe. Rational Krylov algorithms for nonsymmetric eigenvalue problems, II: Matrix pairs. *Linear Algebra and Its Applications*, 197,198:283–295, 1994.

[69] A. Ruhe. Rational Krylov, a practical algorithm for large sparse nonsymmetric matrix pencils. Technical Report UCB/CSD-95-871 (revised), Computer Science Division(EECS), University of California Berkeley, CA 94720, 1995.

[70] Y. Saad. On the rates of convergence of the Lanczos and the block Lanczos methods. *SIAM Journal of Numerical Analysis*, 17(5):687–706, October 1980.

[71] Y. Saad. Projection methods for solving large sparse eigenvalue problems. In B. Kågström and A. Ruhe, editors, *Matrix Pencil Proceedings*, volume 973 of *Lecture Notes in Mathematics*, pages 121–144, Berlin, 1982. Springer–Verlag.

[72] Y. Saad. Iterative solution of indefinite symmetric linear systems by methods using orthogonal polynomials over two disjoint intervals. *SIAM J. Numer. Anal.*, 20:784–810, 1983.

[73] Y. Saad. Chebyshev acceleration techniques for solving nonsymmetric eigenvalue problems. *Mathematics of Computation*, 42:567–588, 1984.

[74] Y. Saad. Practical use of polynomial preconditionings for the conjugate gradient method. *SIAM J. Sci. Statis. Comput.*, 6:865–881, 1985.

[75] Y. Saad. *Numerical Methods for Large Eigenvalue Problems.* Halsted Press, 1992.

[76] Y. Saad and M. H. Schultz. GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7:856–869, 1986.

[77] J. A. Scott. An Arnoldi code for computing selected eigenvalues of sparse real unsymmetric matrices. *ACM Transactions on Mathematical Software*, 21:432–475, 1995.

[78] I. W. Selesnick and C. S. Burrus. Exchange algorithms for the design of linear phase FIR filters and differentiators having flat monotonic passbands and equiripple stopbands. Technical report, Dept. of Electrical and Computer Engineering, Rice University, 1996. To appear in IEEE Trans on Circuits and Systems, Part II.

[79] G. L. G. Sleijpen and H.A. Van der Vorst. A Jacobi-Davidson iteration method for linear eigenvalue problems. *SIAM Journal on Matrix Analysis and Applications*, 17(2):401–425, April 1996.

[80] G.L.G. Sleijpen, J.G.L. Booten, D.R. Fokkema, and H.A. Van der Vorst. Jacobi-Davidson type methods for generalized eigenproblems and polynomial eigenproblems. *BIT*, 36(3):595–633, 1996.

[81] D. C. Sorensen. Implicit application of polynomial filters in a k-step Arnoldi method. *SIAM Journal on Matrix Analysis and Applications*, 13(1):357–385, January 1992.

[82] D. C. Sorensen. Truncated QZ methods for large scale generalized eigenvalue problems. In preparation, Department of Computational & Applied Mathematics, Rice University, Houston, TX 77005, 1997.

[83] D. C. Sorensen, P. A. Vu, and Z. Tomasic. Algorithms and software for large scale eigenproblems on high performance computers. In Adrian Tentner, editor, *Proceedings 1993 Simulation Muticonference*, pages 149–154. Society for Computer Simulation, 1994.

[84] D.C. Sorensen and C. Yang. A truncated RQ-iteration for large scale eigenvalue calculations. Technical Report TR96-06, Department of Computational & Applied Mathematics, Rice Univeristy, Houston, TX 77005, 1996. To appear in SIAM Journal on Matrix Analysis and Applications.

[85] E. L. Stiefel. Kernel polynomials in linear algebra and their numerical applications. *U.S. Nat. Bur. Standards, Appl. Math. Ser.*, 49:1–22, 1958.

[86] G. Szegö. *Orthogonal Polynomials*. American Mathematical Society, New York, 1959.

[87] R. A. Tapia and D. L. Whitley. The projected Newton method has order $1+\sqrt{2}$ for the symmetric eigenvalue problem. *SIAM Journal on Numerical Analysis*, 25(6):1376–1382, 1988.

[88] H. Unger. Nichtlineare Behandlung von Eigenwertaufgaben. *Z. Angew. Math. Mech.*, 30:281–282, 1950.

[89] D. S. Watkins. Some perspectives on the eigenvalue problem. *SIAM Review*, 35(3):430–471, September 1993.

[90] C. Yang, D. C. Sorensen, D. I. Meiron, and B. Wedeman. Numerical computation of the linear stability of the diffusion model for crystal growth simulation. Technical Report CRPC-TR96645-S, Center for Research on Parallel Computation, Rice University, Houston, TX 77251, 1996.

Appendix

Double-shift TRQ for Real Nonsymmetric Matrices

A real nonsymmetric matrix may have eigenvalues that are complex conjugate pairs. The single shift selection strategy discussed above will produce complex matrices H_k and V_k in the TRQ iteration. However, it is usually desirable to work within real arithmetic just as in the standard double shift QR algorithm [22, 23]. To avoid the complex arithmetic introduced by a complex shift, one may choose to work with $\hat{A} = (A - \bar{\mu}I)(A - \mu I)$ rather than $A - \mu I$ alone. It follows from the full Hessenberg reduction $AV = VH$ that $\hat{A}V = V\hat{H}$, where $\hat{H} = (H - \bar{\mu}I)(H - \mu I)$. Clearly, both $\hat{A}$ and $\hat{H}$ are real. We let $\hat{H} = RQ$ be an RQ factorization of $\hat{H}$, and postmultiply $AV = VH$ by Q^H to get $AV_+ = V_+H_+$, where $V_+ = VQ^H$ and $H_+ = QHQ^H$. It is easy to see that V_+ and V are also related by $\hat{A}V_+ = VR$. The first column of V_+ satisfies an inverse power relation with respect to $\hat{A}$ and the first column of V. It is well known that the update of the Hessenberg matrix $H_+ = QHQ^H$ can be done implicitly through a bulge chase process that keeps H_+ in an upper Hessenberg form.

In the large scale setting, we would like to develop the a truncated form of bulge chase process associated with a double shift application. Again, our motivation is to update and maintain only the leading portion of $AV_+ = V_+H_+$. To be more specific, we put $V = (V_k, \hat{V})$, and let

$$\hat{H} = \begin{pmatrix} \hat{H}_{11} & \hat{H}_{12} \\ \hat{H}_{21} & \hat{H}_{22} \end{pmatrix}$$

be partitioned conformally so that

$$\hat{A}(V_k, \hat{V}) = (V_k, \hat{V}) \begin{pmatrix} \hat{H}_{11} & \hat{H}_{12} \\ \hat{H}_{12} & \hat{H}_{22} \end{pmatrix} \tag{7.3}$$

Note that $\hat{H}$ is no longer upper Hessenberg. It has two subdiagonals. The RQ factorization of $\hat{H}$ can be produced by applying a sequence of Householder reflectors from the right to left. The first $n-k-2$ reflectors Q_1, Q_2,...,Q_{n-k-2} effect only the last $n-k$ columns of H and V. Since these columns will not be retained in the truncated version of the double shifted RQ algorithm, we do not need to know Q_1, Q_2,...,Q_{n-k-2} explicitly. However, after applying these reflectors, a set of well defined linear equations emerge. The solutions of these equations help to determine the remaining reflectors needed to construct the first k columns of V_+ and H_+. In the following, we discuss how these equations are derived and how the partial RQ update may be completed without forming Q_1, Q_2,...,Q_{n-k-2}.

Suppose $\hat{H}$ has been partially factored such that

$$\hat{H} = \begin{pmatrix} \hat{H}_{11} & \hat{M} \\ \hat{H}_{21} & \hat{R} \end{pmatrix} \begin{pmatrix} I_k & 0 \\ 0 & \hat{Q} \end{pmatrix},$$

where $\hat{Q}$ is the accumulation of the first $n-k-2$ reflectors, $\hat{R} = \begin{pmatrix} \gamma_{11} & \gamma_{12} & r_1^T \\ \gamma_{21} & \gamma_{22} & r_2^T \\ 0 & 0 & \hat{R}_3 \end{pmatrix}$, and $\hat{R}_3$ is an $(n-k-2) \times (n-k-2)$ upper triangular matrix. Multiplying (7.3) from the right by $\begin{pmatrix} I_k & 0 \\ 0 & \hat{Q}^H \end{pmatrix}$ and equating the first $k+2$ columns give rise to the equation

$$\hat{A}(V_k, v_{k+1}^+, v_{k+2}^+) = (V_k, v_{k+1}, v_{k+2}) \begin{pmatrix} \hat{H}_{11} & g_1 & g_2 \\ \hat{h}_{k+1}^T & \gamma_{11} & \gamma_{12} \\ \hat{h}_{k+2}^T & \gamma_{21} & \gamma_{22} \end{pmatrix}, \tag{7.4}$$

where

$$v_{k+j}^{+} = \hat{V}\hat{Q}^H e_j, \quad \hat{h}_{k+i}^T = e_i^T \hat{H}_{21}, \quad g_j = \hat{M} e_j, \quad \text{and } \gamma_{ij} = e_i^T \hat{R} e_j, \quad i,j = 1,2.$$

The same sequence of reflectors may be used to partially update the Hessenberg reduction form $AV = VH$ such that

$$A(V_k, \hat{V}\hat{Q}^H) = (V_k, \hat{V}\hat{Q}^H) \begin{pmatrix} H_k & M\hat{Q}^H \\ \beta_k \hat{Q} e_1 e_k^T & \hat{Q} H_{n-k} \hat{Q}^H \end{pmatrix}, \tag{7.5}$$

where

$$\beta_k \hat{Q} e_1 = (\theta_{11},\, \theta_{21},\, \theta_{31}, 0, ..., 0)^T, \quad \hat{Q} H_{n-k} \hat{Q}^T = \begin{pmatrix} \theta_{12} & \theta_{13} & c_1^T \\ \theta_{22} & \theta_{23} & c_2^T \\ \theta_{32} & \theta_{33} & c_3^T \\ 0 & 0 & C_4 \end{pmatrix},$$

and the trailing submatrix C_4 is upper Hessenberg. At this point, the first $k+3$ columns of (7.5) satisfy

$$A(V_k, v_{k+1}^{+}, v_{k+2}^{+}, v_{k+3}^{+}) = (V_k, v_{k+1}^{+}, v_{k+2}^{+}, v_{k+3}^{+}) \begin{pmatrix} H_k & h_1 & h_2 \\ \theta_{11} e_k^T & \theta_{12} & \theta_{13} \\ \theta_{21} e_k^T & \theta_{22} & \theta_{23} \\ \theta_{31} e_k^T & \theta_{32} & \theta_{33} \end{pmatrix}, \tag{7.6}$$

where $h_j = M\hat{Q}^H e_j$ and $v_{k+j}^{+} = v_{k+1}\hat{Q}^H$ $(j = 1,2,3.)$ Assuming that v_{k+i}^{+} and θ_{ij} are available, we may complete the RQ factorization by first constructing a reflector Q_1 to annihilate the off diagonal entries on the last row of

$$\check{H} = \begin{pmatrix} H_k & h_1 & h_2 \\ \theta_{11} e_k^T & \theta_{12} & \theta_{13} \\ \theta_{21} e_k^T & \theta_{22} & \theta_{23} \\ \theta_{31} e_k^T & \theta_{32} & \theta_{33} \end{pmatrix},$$

and then use the bulge chase scheme to keep the leading $(k+1) \times (k+1)$ portion of $\check{H}$ in an upper Hessenberg form. It may be verified that Q_1 also zeros out the off-diagonal entries on the last row of $\tilde{H} = \begin{pmatrix} \hat{H}_{11} & g_1 & g_2 \\ \hat{h}_{k+1}^T & \gamma_{11} & \gamma_{12} \\ \hat{h}_{k+2}^T & \gamma_{21} & \gamma_{22} \end{pmatrix}$, thus it may be constructed from $\hat{h}_{k+2}^T e_k$, γ_{21} and γ_{22} directly. Hence v_{k+3}^+ and θ_{3j} do not need to be determined explicitly. It remains to show how v_{k+i}^+ and θ_{ij} $(i = 1, 2,\ j = 1, 2, 3)$ may be determined. Let

$$\hat{U} = (v_{k+1}, v_{k+2}),\quad \hat{U}^+ = (v_{k+1}^+, v_{k+2}^+),\quad G = (g_1,\ \ g_2),\ \text{and}\ \Gamma = \begin{pmatrix} \gamma_{11} & \gamma_{12} \\ \gamma_{21} & \gamma_{22} \end{pmatrix}.$$

The equation (7.4) and the condition $V_k^H \hat{U} = 0$ may be expressed by the equation:

$$\begin{pmatrix} \hat{A} & V_k \\ V_k^H & 0 \end{pmatrix} \begin{pmatrix} \hat{U}^+ \\ -G \end{pmatrix} = \begin{pmatrix} \hat{U}\Gamma \\ 0 \end{pmatrix}.$$

If Γ is nonsingular, we may solve $\hat{U}^+$ and Γ from the above equation as follows.

1. Solve $\begin{pmatrix} \hat{A} & V_k \\ V_k^H & 0 \end{pmatrix} \begin{pmatrix} W \\ Z \end{pmatrix} = \begin{pmatrix} \hat{U} \\ 0 \end{pmatrix}$.
2. QR factor $W = \hat{U}^+ R$, where $(\hat{U}^+)^H \hat{U}^+ = I$, and R is upper triangular.
3. $\Gamma \leftarrow R^{-1}$ (This matrix is never actually used or computed).

Once $\hat{U}^+$ is available, the values of h_i and θ_{ij} $(i = 1, 2,\ j = 1, 2, 3)$ are determined by (7.6). They can be computed by

$$h_i = V_k^H A v_{k+i}^+,\quad \text{and}\quad \theta_{ij} = v_{k+i}^H A v_{k+j-1}^+.$$

In the meantime, a reflector Q_1 may be generated such that

$$(0, ..., \xi, \gamma_{21}, \gamma_{22}) Q_1^H = (0, ..., \tau),$$

where $\xi = \hat{h}_{k+2}^T e_k$ and $\tau = \sqrt{\xi^2 + \gamma_{12}^2 + \gamma_{22}^2}$. This reflector is used to initiate the bulge chase process that completes the update of H_k.

The bulge chase process starts from applying Q_1^H from the right and Q_1 from the left to the matrix

$$G = \begin{pmatrix} H_k & h_1 & h_2 \\ \theta_{11} e_k^T & \theta_{12} & \theta_{13} \\ \theta_{21} e_k^T & \theta_{22} & \theta_{23} \end{pmatrix}$$

The next reflector Q_2 is constructed to annihilate the $(k+2, k-1)$ and $(k+2, k)$ entries of G. Subsequent reflectors are generated and applied to G such that bulges introduced by the previous reflector is pushed towards the upper left corner of H. We summarize the above discussion in Algorithm 5.

Algorithm 5: (DBTRQ) Doubly shifted Truncated RQ-iteration

Input: $(A, V_{k+1}, H_{k+1}, f_{k+1})$ with $AV_{k+1} = V_{k+1}H_{k+1} + f_{k+1}e_{k+1}^T$, $V_{k+1}^H V_{k+1} = I$, H_k upper Hessenberg.
Output: (V_k, H_k) such that $AV_k = VH_k, V_k^H V_k = I$

1. Put $\beta_{k+1} = \|f_{k+1}\|$, $\beta_k = H_{(k+1,k)}$; $v_{k+2} \leftarrow f_{k+1}/\beta_{k+1}$; $v_{k+1} \leftarrow V_{k+1}e_{k+1}$;
2. for $j = 1, 2, 3, ...$ until *convergence*,
  2.1. Select a shift $\mu \leftarrow \mu_j$;
  2.2. Put $\hat{A} = (A - \bar{\mu}I)(A - \mu I)$, $\hat{U} \leftarrow (v_{k+1},\ v_{k+2})$;
  2.3. Solve $\begin{pmatrix} \hat{A} & V_k \\ V_k^H & 0 \end{pmatrix} \begin{pmatrix} W \\ Z \end{pmatrix} = \begin{pmatrix} \hat{U} \\ 0 \end{pmatrix}$;
  2.4. QR factor W such that $W = \hat{U}^+ R$, $(\hat{U}^+)^H \hat{U}^+ = I$ and R is upper triangular.
  2.5. $\Gamma \leftarrow R^{-1}$; $h_i \leftarrow V_k^H A v_{k+i}$; $\theta_{i1} \leftarrow v_{k+i}^H A v_k$; $\theta_{i,j+1} \leftarrow v_{k+i}^H A v_k$;
  2.7. Put $\check{H} = \begin{pmatrix} H_k & h_1 & h_2 \\ \theta_{11}e_k^T & \theta_{12} & \theta_{12} \\ \theta_{21}e_k^T & \theta_{22} & \theta_{23} \end{pmatrix}$;
  2.8. Let $\xi = \beta_k\beta_{k+1}$; Construct a reflector Q_1 such that $(\xi,\ \gamma_{21},\ \gamma_{22})Q_1^H = (0,\ 0,\ \tau)$;
  2.9. Put $\check{H} \leftarrow Q_1 G Q_1^H$; $(V_k, v_{k+1}^+, v_{k+2}^+) \leftarrow (V_k, v_{k+1}^+, v_{k+2}^+)Q_1^H$;
  2.10. Bulge chase $H \leftarrow Q_{k-2}Q_{k-3}\cdots Q_2 \check{H} Q_2^H \cdots Q_{k-3}^H Q_{k-2}^H$; $(V_k,\ v_{k+1},\ v_{k+2}) \leftarrow (V_k, v_{k+1}^+,\ v_{k+2}^+)Q_2^H \cdots Q_{k-2}^H$;
  2.11. Put $\beta_k = e_{k+1}^T H_{k+1} e_k$; $f_{k+1} \leftarrow Av_{k+1}$; $f_{k+1} \leftarrow f_{k+1} - V_{k+1}^H f_{k+1}$; $\beta_{k+1} \leftarrow \|f_{k+1}\|$;
3. end;

Figure 7.2 Doubly shifted TRQ iteration