October 2018

Inexact and Nonlinear Extensions of the FEAST Eigenvalue Algorithm

Brendan E. Gavin
University of Massachusetts - Amherst

INEXACT AND NONLINEAR EXTENSIONS OF THE FEAST EIGENVALUE ALGORITHM

A Dissertation Presented

by

BRENDAN GAVIN

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 2018

Electrical and Computer Engineering

INEXACT AND NONLINEAR EXTENSIONS OF THE FEAST EIGENVALUE ALGORITHM

A Dissertation Presented

by

BRENDAN GAVIN

Approved as to style and content by:

Eric Polizzi, Chair

Zlatan Aksamija, Member

Blair Perot, Member

Yousef Saad, Member

Christopher Hollot, Department Head
Electrical and Computer Engineering

DEDICATION

To my family, whose steady support made this possible.

ABSTRACT

INEXACT AND NONLINEAR EXTENSIONS OF THE FEAST EIGENVALUE ALGORITHM

SEPTEMBER 2018

BRENDAN GAVIN

B.Sc., UNIVERSITY OF MASSACHUSETTS AMHERST

M.Sc., UNIVERSITY OF MASSACHUSETTS AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Eric Polizzi

Eigenvalue problems are a basic element of linear algebra that have a wide variety of applications. Common examples include determining the stability of dynamical systems, performing dimensionality reduction on large data sets, and predicting the physical properties of nanoscopic objects. Many applications require solving large dimensional eigenvalue problems, which can be very challenging when the required number of eigenvalues and eigenvectors is also large. The FEAST algorithm is a method of solving eigenvalue problems that allows one to calculate large numbers of eigenvalue/eigenvector pairs by using contour integration in the complex plane to divide the large number of desired pairs into many small groups; these small groups of eigenvalue/eigenvector pairs may then be simultaneously calculated independently of each other. This makes it possible to quickly solve eigenvalue problems that might otherwise be very difficult to solve efficiently.

The standard FEAST algorithm can only be used to solve eigenvalue problems that are linear, and whose matrices are small enough to be factorized efficiently (thus allowing linear systems of equations to be solved exactly). This limits the size and the scope of the problems to which the FEAST algorithm may be applied. This dissertation describes extensions of the standard FEAST algorithm that allow it to efficiently solve nonlinear eigenvalue problems, and eigenvalue problems whose matrices are large enough that linear systems of equations can only be solved inexactly.

TABLE OF CONTENTS

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

This dissertation is a description of two algorithms for solving eigenvalue problems. One algorithm, referred to here as IFEAST, is for solving linear eigenvalue problems when one is required, often by limitations in computing hardware, to use only matrix-vector multiplication as the primary computational tool. The other algorithm, referred to here as NLFEAST, is for solving nonlinear eigenvalue problems. As their names would seem to imply, both of these algorithms are modifications of a third algorithm, called FEAST, which solves linear eigenvalue problems using matrix factorization as the primary computational tool.

The difference between IFEAST and NLFEAST is somewhat artificial, and has more to do with their intended use cases than with the underlying mechanisms by which they work. One can, for example, implement NLFEAST in such a way that it uses only matrix-vector multiplication, with the end result being that it can solve many of the same kinds of problems that IFEAST can. Nonlinear eigenvalue problems tend to be a subject of narrower and more specialized interest, however, and the assumption of linearity makes the analysis of algorithms like IFEAST much easier. This dissertation will thus generally treat the two separately, even though linear eigenvalue problems are actually a subset of nonlinear eigenvalue problems.

The rest of this introduction will discuss the context for eigenvalue problems and why, specifically, they are of interest to a scientist or engineer. Chapter 2 will discuss some of the mathematical background behind how eigenvalue problems are usually solved, as well as some of the background behind how linear systems of equations

are usually solved (because, as we will see, the two are often intimately related). It will also discuss the motivation for this work in more detail. The standard FEAST algorithm, the IFEAST algorithm, and the NLFEAST algorithm will be covered in Chapters 3, 4, and 5 respectively.

1.1 Why eigenvalue problems?

An "eigenvalue problem" consists of finding the eigenvalues and eigenvectors of a matrix[1]. The eigenvalues and eigenvectors of a matrix A are scalar values λ and vectors x such that

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

The terms "eigenvector" and "eigenvalue" are originally German, and perhaps the best English translations are "characteristic vectors" and "characteristic values". They are what define a matrix, in the sense that every diagonalizable matrix can be written in terms of its eigenvalues and eigenvectors as

$$A = X\Lambda X^{-1} \tag{1.2}$$

where X is a matrix whose column vectors are the eigenvectors x, and Λ is a diagonal matrix whose diagonal entries are the corresponding eigenvalues. This is known as an eigenvalue decomposition.

1.1.1 Linear Time Dependent Systems of Equations

Being able to solve Equation (1.1), or calculate matrix decompositions like in Equation (1.2), is especially useful when dealing with systems of linear, time dependent equations, e.g.

[1]Or, more accurately, a matrix pencil; more on that in Chapter 2.

$$\begin{aligned}
\frac{d}{dt}v_1(t) &= A_{11}v_1(t) + A_{12}v_2(t) + A_{13}v_3(t) \\
\frac{d}{dt}v_2(t) &= A_{21}v_1(t) + A_{22}v_2(t) + A_{23}v_3(t) \\
\frac{d}{dt}v_3(t) &= A_{31}v_1(t) + A_{32}v_2(t) + A_{33}v_3(t)
\end{aligned} \tag{1.3}$$

Systems of differential equations like this can be concisely written in terms of vectors and matrices, where the functions $v_1(t)$, $v_2(t)$, and $v_3(t)$ can be collected into a vector, and the coefficients $A_{11}...A_{33}$ can be collected into a matrix:

$$v(t) = \begin{bmatrix} v_1(t) \\ v_2(t) \\ v_3(t) \end{bmatrix} \qquad A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \tag{1.4}$$

Equation (1.3) then becomes

$$\frac{d}{dt}v(t) = Av(t), \tag{1.5}$$

and the solution, given some initial condition $v(t_0)$, is

$$v(t) = Xe^{\Lambda t}X^{-1}v(t_0), \tag{1.6}$$

where $e^{\Lambda t}$ is the diagonal matrix

$$e^{\Lambda t} = \begin{bmatrix} e^{\lambda_1 t} & & \\ & e^{\lambda_2 t} & \\ & & e^{\lambda_3 t} \end{bmatrix} \tag{1.7}$$

Equations like (1.5) occur often in engineering. This is especially true of electrical engineering; the behavior of any electrical circuit that consists entirely of linear circuit elements, such as resistors, capacitors, inductors, and linear amplifiers, can be described by a system of differential equations that takes the form of Equation (1.5).

An eigenvalue decomposition might be used for solving Equation (1.5) if the number of equations is small enough that the eigenvalue decomposition can be calculated easily, or if very high precision is required for the solution.

Even if the number of equations is too large for the full eigenvalue decomposition to be practically calculated[2], a partial eigenvalue decomposition - calculating only some of the eigenvalues and eigenvectors - can still yield useful information. The behavior of solutions to Equation (1.5) are governed in part by the eigenvalues λ_i; eigenvalues with positive real parts indicate that there are solutions whose magnitudes grow exponentially with time, and eigenvalues with nonzero imaginary parts indicate solutions that will oscillate. This kind of information can be obtained by calculating small numbers of eigenvalues in specific regions of the complex plane, rather than by finding all of the eigenvalues and eigenvectors at once.

1.1.2 Quantum Mechanics

Eigenvalues and eigenvectors play a central role in quantum mechanics. The behavior of quantum mechanical systems is governed by the time dependent Schrödinger equation,

$$\frac{d}{dt}\psi(x,t) = -\frac{i}{\hbar}\hat{H}\psi(x,t), \tag{1.8}$$

where $\psi(x,t)$ is called the wavefunction, and $\hat{H}$ is the Hamiltonian operator, which is determined by the physical system under consideration. For example, the Hamiltonian operator for a quantum harmonic oscillator is $\hat{H} = \frac{-\hbar^2}{2m}\frac{d^2}{dx^2} + \frac{1}{2}kx^2$, where m is the mass of the oscillator, and k is the force constant.

Measurable quantities are represented in quantum mechanics by Hermitian operators [75]; the Hamiltonian operator in Equation (1.8), for example, represents energy. If $\hat{M}$ is an operator corresponding to some measurable quantity, then the possible

[2]E.g. the number of equations is in the many tens of thousands, or more, and the computer being used for the calculations is a desktop computer.

outcomes of the measurement corresponding to $\hat{M}$ are its eigenvalues λ_i, and the probability of measuring a particular value λ_i at time t is equal to $(m_i(x), \psi(x,t))^2$, which is square of the inner product of the corresponding eigenvector m_i and the wavefunction ψ at time t. Determining the physical properties of objects whose behavior is governed by quantum mechanics, and predicting the outcomes of physical measurements of such objects, thus amounts to solving the eigenvalue problems. The most prominent of these eigenvalue problems is

$$-\frac{i}{\hbar}\hat{H}\psi_i(x) = E_i\psi_i(x). \tag{1.9}$$

This is called the time-independent Schrödinger equation, because the probability distributions that are associated with its solution vectors $\psi_i(x)$ do not change change with time.

Eigenvalue calculations for predicting the properties of quantum systems have become increasingly important in modern scientific and engineering practice. Microelectronic devices are now constructed at sufficiently small length scales that accurate models of device behavior require the use of full quantum-mechanical calculations in order to estimate quantities of interest such as frequency response and band structure. Examples of such devices include transistors based on quantum dots, carbon nanotubes, or 2D nanomaterials. Eigenvalue problems from quantum mechanics also play a large role in the chemical and biological sciences, where they can be used to predict chemical properties and reaction rates.

The lowest energy quantum states are the ones that are most likely to be occupied at standard temperatures and pressures, and so most of the important properties of physical matter in every-day life are dominated by the properties of the lowest-energy physical states [61]. For this reason particular emphasis is often placed on examining the lowest eigenvalues and corresponding eigenvectors of the Hamiltonian operator $\hat{H}$. In practice it is usually impossible to calculate the eigenvalues and eigenvectors of

arbitrary differential operators, and so the original Schrödinger equation from (1.8) is replaced with a discretized version in which the Hamiltonian operator $\hat{H}$ becomes a matrix approximation, and the wavefunction $\psi(x,t)$ becomes by a vector approximation in which the space-dependence is discretized:

$$\frac{d}{dt}\psi(x,t) = -\frac{i}{\hbar}\hat{H}\psi(x,t) \quad \longrightarrow \quad \frac{d}{dt}v(t) = -\frac{i}{\hbar}Av(t) \tag{1.10}$$

The end result is essentially the same equation from the previous section, Equation (1.5). The method of discretization is often chosen so that most of the entries in the matrix A are zero (and therefore do not need to be stored), or so that none of the coordinates of A need to be stored at all, with the matrix instead being represented by an efficient computer routine for calculating matrix-vector products. Representing A using such efficient storage schemes makes it possible to generate large, highly-accurate discretizations of complicated physical systems.

Much of the computational work in quantum mechanics applications consists of solving the discretized, time-independent Schrödinger equation $-\frac{i}{\hbar}Ax = \lambda x$ in order to identify the stationary states and corresponding energies of a quantum system. A popular method of approximation, Density Functional Theory (DFT) [38, 44, 69], involves solving a large number such eigenvalue problems in order to efficiently calculate the properties of many-particle systems. Rather than calculating the solution of a single linear eigenvalue problem, as one does in single-particle quantum mechanics, DFT solves a sequence of many linear eigenvalue problems in order to converge to the solution for a single nonlinear eigenvector problem, with the solution of that nonlinear eigenvalue problem giving the ground state charge density for a system of multiple interacting particles. The challenge of solving eigenvalue problems like these is what motivates much the research that is described in this dissertation.

1.1.3 Dimensionality Reduction for Data

Data points from measurements consisting of n scalar values can be represented vectors in an n-dimensional vector space. Images, for example, can be represented as vectors in which each coordinate is the value of a pixel, or text documents can be represented as vectors in which each coordinate is the number of times that a particular word appears in a given document. Although the original data is embedded in a linear vector space of dimension n, in many cases the data points themselves may exist primarily on some manifold of dimension k that is substantially smaller [14] than the original dimension n [3]. If a representation for the data on this k-dimensional manifold can be calculated, then its salient features can be systematically identified more easily. The process of calculating an accurate representation of a collection of data using some a lower-dimensional space is called "dimensionality reduction".

Several common dimensionality reduction methods can be implemented by solving eigenvalue problems. One of the most common is the Singular Value Decomposition (or SVD). If a data set is collected into a matrix A such that the coordinates of each data point a_i forms a column vector of A, i.e.

$$A = [a_1 \;\; a_2 \;\; a_3 \;\; ... \;\; a_m], \tag{1.11}$$

then the data matrix A can be decomposed using the SVD:

$$A = U\Sigma V^H. \tag{1.12}$$

U is an $n \times \min(n, m)$ orthogonal matrix, V is an $m \times \min(n, m)$ orthogonal matrix, and Σ is a $\min(n, m) \times \min(n, m)$ diagonal matrix whose diagonal entries are positive numbers. The diagonal entries of Σ are called the "singular values" of A. The matrix

[3] The assumption that this is true is usually referred to as the "manifold hypothesis".

A can be approximated by replacing Σ with a different diagonal matrix of dimension $k \times k$, $\tilde{\Sigma}$, whose diagonal entries are the k largest-magnitude singular values:

$$A \approx \tilde{A} = U\tilde{\Sigma}V^H. \tag{1.13}$$

If the k largest-magnitude singular values of A are substantially larger than the remaining singular values are, then $\tilde{A}$ will be a good approximation for A, and so the column vectors of $\tilde{A}$ will be good representation of the original data points in a k-dimensional hyperplane. Perhaps the most common application in which SVD is used for dimensionality reduction is Principle Components Analysis (PCA), wherein a collection of data points is centered so that their sample mean is the zero vector before their SVD is calculated [14].

One way to calculate Singular Value Decompositions is by solving eigenvalue problems. One method finds V and Σ by calculating the eigenvalue decomposition of A^HA, and then calculates U using V and Σ:

$$A^HA = V\Sigma^2V^H, \quad U = AV\Sigma^{-1}. \tag{1.14}$$

Another calculates U, V, and Σ by calculating the eigenvalue decomposition of an augmented matrix:

$$\begin{bmatrix} 0 & A \\ A^H & 0 \end{bmatrix} = \begin{bmatrix} -U & -U \\ -V & V \end{bmatrix} \begin{bmatrix} -\frac{1}{2}\Sigma & 0 \\ 0 & \frac{1}{2}\Sigma \end{bmatrix} \begin{bmatrix} U^H & V^H \\ U^H & -V^H \end{bmatrix} = X\Lambda X^{-1} \tag{1.15}$$

Which of these methods is most advantageous depends on the problem at hand and on the algorithm that is being used to solved eigenvalue problems.

Eigenvalue problems can also be used for finding representations of data in nonlinear manifolds, rather than linear hyperplanes. Kernel PCA [80] is a version of

Principle Components Analysis that applies a nonlinear transformation (called a kernel function) to a collection of data points before doing PCA. Algorithms like Laplacian Eigenmap [5], Locally Linear Embedding [65], and Isomap [95] use the distances between data points in a data collection in order to form a graph that approximates the manifold in which the data is assumed to be embedded, and then find optimized graph embeddings by solving eigenvalue problems. Implementing any of these methods for large data sets will usually require the use of eigenvalue algorithms that need only matrix-vector multiplication for performing computations.

1.2 Why FEAST?

The challenge with regards to eigenvalue problems in the context of modern computing is to be able to use computational resources efficiently in order to solve eigenvalue problems that are exceptionally large. Several algorithms already exist for solving solving large eigenvalue problems, and there are a variety of widely-available software packages that implement these algorithms efficiently [4, 11, 37, 43, 49, 89]. Section 2.3 in Chapter 2 discusses several of these algorithms in some detail.

These algorithms do not, however, scale to larger problem sizes in a way that that is synergistic with modern computing architecture design. The modern design paradigm for building large, powerful computers is to connect a large number of smaller, less powerful computers together in a network, and then use all of these networked computers together to to solve problems. Sometimes this takes the form of large data centers that are filled with discrete computers that are connected together in an actual network. Other times it takes the form of specialized coprocessors that consist of a large number of computing cores that are all implemented on a single integrated circuit. In many cases both of these design paradigms will be used at the same time, in addition to others.

Although increasing computing power through increased parallelism can be achieved in a number of different ways, every design paradigm that relies on parallel computation in order to achieve high computing power shares the same central constraint: in order to use parallelism efficiently, it must be possible to break up a large problem into a collection of smaller tasks that can all be performed independently of each other. Most algorithms for solving eigenvalue problems violate this constraint at a mathematical level; even if they can be implemented in a way that allows them to be run on computers that use distributed memory and processing, they still require a sequential series of operations that can not be performed independently of each other.

Much of the contemporary research into solving eigenvalue problems consists of developing algorithms that can take advantage of parallel computing at a mathematical level. These algorithms can solve eigenvalue problems in such a way that the original problem to be solved is divided into separate tasks that can all be performed independently of each other, with the intention that the computational resources that are available on parallel computing platforms can be used in a maximally efficient manner. The FEAST algorithm is one example of such an algorithm, and the work described in this dissertation consists of extending the FEAST algorithm to be applicable to new or more difficult problem domains in order to better take advantage of the parallelism that is available in modern computing architectures.

1.3 Novel Contributions

This dissertation makes the following novel contributions to the study of numerical algorithms:

Basic and Generalized IFEAST algorithms: I show that the FEAST algorithm can be efficiently implemented for matrices that can not be factorized, and provide theory and analysis that explain the properties of the resulting IFEAST algorithms. In the context of the existing literature, this is equivalent to showing

how to extend Inexact Shift and Invert Iteration to use multiple shifts in the complex plane. This is the subject of Chapter 4.

Equivalence of Basic IFEAST and Krylov methods: I show that there is an intimate relationship between the Basic IFEAST algorithm and Krylov methods for solving eigenvalue problems. When Basic IFEAST is implemented with Krylov linear system solvers, it becomes a Krylov eigenvalue solver itself. For certain linear system algorithms it is possible to show an exact equivalence between Basic IFEAST and block versions of well-known Krylov eigenvalue algorithms. This is discussed in Section 4.5.

Interpretation of IFEAST as a Polynomial Filter: Section 4.4 shows explicitly that IFEAST (and therefore, by extension, Inexact Shift and Invert Iteration) is a polynomial eigenvalue filter algorithm, and that it has properties that distinguish it from conventional methods for building polynomial filters. Of particular interest is the fact that it fits a polynomial filter only at the eigenvalues of the matrix that is being filtered.

Nonlinear FEAST algorithm: Chapter 5 shows that the same modification that allows IFEAST to efficiently solve generalized, linear eigenvalue problems can be extended further to allow the FEAST algorithm to solve general, nonlinear eigenvalue problems. This amounts to developing an extension of Residual Inverse Iteration that uses multiple shifts to enhance convergence.

CHAPTER 2

MATHEMATICAL BACKGROUND

The FEAST eigenvalue algorithm, and the variations of it that are described in this dissertation, is closely related to several other well-known algorithms for solving both eigenvalue problems and linear systems of equations. The following sections describe the mathematical background for these other algorithms in order to contextualize and clarify the original results that are reported in later chapters.

These sections will describe algorithms that all operate under a particular constraint: the only operation that they use for performing calculations with large matrices is matrix-vector multiplication. It is assumed that it is difficult, inefficient, or even impossible to calculate factorizations or decompositions of large matrices, and indeed this is taken to be the operational definition of a "large" matrix. Matrices that are large in this sense are not stored as full, two-dimensional arrays of coordinates. Instead they are stored either as "sparse" matrices, in which only the nonzero coordinates of the matrix are stored, or they are stored implicitly by defining an efficient computer routine for performing matrix-vector multiplications.

It is assumed that any matrix operation is both possible and efficient for a "small" matrix, where small matrices are defined to be those whose coordinates can be stored in full. Small matrices in this sense do not necessarily have a small dimension, they are just small enough that it is possible to perform operations such as QR or Singular Value Decompositions on them. An example of such matrices that will occur frequently in this chapter are tall rectangular matrices that have a relatively small

number of columns, but which may have a number of rows that is equal to the dimension of another, "large" matrix on which calculations are being performed.

The algorithms that will be discussed in the following sections are basic versions of selected algorithms that have some important and explicable relation to FEAST and its variants. The full range of possible and known algorithms for solving eigenvalue problems and linear systems of equations is immense, and the literature on it is voluminous. Entire books can be, and have been, written on these subjects without necessarily being comprehensive. Where appropriate the reader will be directed to references in the literature for a more detailed and comprehensive coverage of the subjects under consideration. For general references on these topics, I recommend the following books: [60, 71, 73, 91, 99, 102].

2.1 Krylov Subspaces

The problems that we will consider in this chapter are eigenvalue problems and linear systems of equations, i.e.

$$Ax = \lambda x \quad \text{and} \quad Ax = b, \tag{2.1}$$

where A is some $n \times n$ matrix, x and b are n-dimensional vectors, λ is a scalar, and we want to calculate the x (and λ) that makes these equations true.

I assume that A is large enough that we can only access it through the operation of matrix-vector multiplication, and I also assume that the number of matrix-vector multiplications that can be performed in a reasonable amount of time is k, which is lower (often substantially lower) than the dimension of A. In this case the Equations (2.1) can only be solved approximately, and so what we actually seek to calculate is some vector $\tilde{x}$ that is sufficiently close to x in some sense.

Because the only operation that we can use with A is matrix-vector multiplication, the approximate solutions $\tilde{x}$ to (2.1) always take the same form,

$$\tilde{x} = p(A)x_0 = \sum_{i=0}^{k} a_i A^i x_0. \tag{2.2}$$

The vector x_0 is some suitable starting vector; for linear systems of equations it is usually chosen to be $x_0 = b$, and for eigenvalue problems x_0 is typically random. The matrix $p(A)$ is the k-degree matrix polynomial

$$p(A) = \sum_{i=0}^{k} a_i A^i. \tag{2.3}$$

It is not always the case that using an approximation like Equation (2.2) is highly effective, or even a particularly good idea; it is simply the only possible method of solution when we constrain ourselves to using only matrix-vector multiplications with A.

Another way of writing Equation (2.2) is to say that $\tilde{x}$ belongs to a Krylov subspace of order k, i.e.

$$\tilde{x} \in \mathcal{K}_k(A, x_0), \qquad \mathcal{K}_k(A, x_0) = span\{x_0, Ax_0, A^2x_0, ..., A^kx_0\}. \tag{2.4}$$

A Krylov subspace of order k for a matrix A and starting vector x_0, denoted $\mathcal{K}_k(A, x_0)$, is simply the subspace that spans the application of all possible degree-k matrix polynomials of A to the vector x_0. A Krylov subspace of order k has dimension $k+1$, provided[1] that $rank(A) \geq (k+1)$ and that x_0 is spanned by at least $k+1$ eigenvectors of A.

Krylov subspace methods operate by forming (either implicitly or explicitly) an $n \times (k+1)$ matrix of basis vectors V for $\mathcal{K}_k(A, x_0)$, which they use to try to efficiently find

[1] I assume, for the purposes of this dissertation, that this will always be true.

approximate solutions $\tilde{x} = V\tilde{x}_v$ to the problems in Equations (2.1). The differences between different Krylov subspace algorithms lie in how, specifically, they form the basis V and use it to approximate a solution. The fundamental mathematics of how the solutions to the problems in (2.1) are best approximated with matrix polynomials do not vary, but there are a variety of different approximations and computational tricks that can be used to achieve good computational performance, depending on the properties of the matrix A.

There are also block Krylov subspaces, which span the application of matrix polynomials to block matrices of column vectors, e.g.

$$\mathcal{K}_k(A, X_0) = span\{X_0, AX_0, A^2X_0, ..., A^kX_0\}, \tag{2.5}$$

where X_0 is an $n \times m$ matrix, usually with $m \ll n$ for the applications considered in this dissertation. A basis for such a block Krylov subspace has dimension $m(k + 1)$. Block Krylov subspaces can be useful for solving linear systems of equations that have multiple right hand sides, i.e. $AX = B$ where B is an $n \times m$ matrix of m right hand sides; with block Krylov subspaces all of the right hand sides can be solved simultaneously. They are also useful - indeed, necessary - when solving eigenvalue problems for matrices whose eigenvalues have multiplicity greater than one. The opportunity to use block Krylov methods arises often in the context of the FEAST algorithm, which naturally operates on blocks of multiple vectors, but it is not always advisable to use block algorithms even when it may seem to be natural or advantageous. This is because (depending on the particular algorithm being used) the memory required for storing a block Krylov basis may be substantial, which can obviate the benefits of using block Krylov subspaces.

The following two subsections describe two common methods for efficiently generating sets of basis vectors for Krylov subspaces.

2.1.1 Arnoldi

Arnoldi iterations are a procedure for building a basis V for the $(k+1)$ dimensional subspace $\mathcal{K}_k(A, b)$ such that $V^H V = I_{(k+1)\times(k+1)}$ [71]. With Arnoldi iterations the first basis vector is assigned to be $v_1 = b/||b||$. Each new basis vector v_i is initially assigned as $v_i = Av_{i-1}$, and then orthogonalized against all of the previous basis vectors before being normalized.

The coefficients that are used to orthonormalize the Arnoldi basis as it is generated can be saved in an upper-Hessenberg matrix that has particular properties. Specifically, if V_k is the Arnoldi basis for $\mathcal{K}_k(A, b)$ and H_k is the upper-Hessenberg produced by the Arnoldi process, then

$$AV_k = V_{k+1}\tilde{H}_k = V_{k+1}(H_k + h_{k+1,k}e_{k+1}e_k^T), \tag{2.6}$$

where e_k is the k^{th} canonical basis vector. The matrix $\tilde{H}_k = (H_k + h_{k+1,k}e_{k+1}e_k^T)$ is the same as the upper-Hessenberg matrix H_k, but with an additional row whose $(k+1)^{\text{th}}$ column element is the $h_{k+1,k}$ coefficient from the Arnoldi process. A result of this relation is that

$$V_k^H AV_k = H_k, \tag{2.7}$$

which will also be important when considering the solution of eigenvalue problems.

The benefit of Arnoldi iterations is that they allow one to calculate an orthonormal basis for $\mathcal{K}_k(A, b)$ in a numerically stable way, and they are the foundation - either implicitly or explicitly - for many methods of solving linear systems of equations.

2.1.2 Lanczos Biorthogonalization

Another option for calculating Krylov subspace bases is Lanczos Biorthogonalization [71] (sometimes called the two-sided Lanczos algorithm, or Non-Hermitian Lanczos), which is often referred to simply as Lanczos. Lanczos operates similarly to Arnoldi but, rather than calculating a single orthogonal set of basis vectors V for

$\mathcal{K}_k(A, b)$, it calculates two biorthogonal sets of basis vectors V and W, with V being a basis for $\mathcal{K}_k(A, b_1)$ and W being a basis for $\mathcal{K}_k(A^H, b_2)$. The basis sets V and W being biorthogonal means that basis vector w_i is orthogonal to basis vector v_j when $i \neq j$, i.e.

$$W^H V = D, \quad D = \begin{bmatrix} w_1^H v_1 & & & \\ & w_2^H v_2 & & \\ & & \ddots & \\ & & & w_{k+1}^H v_{k+1} \end{bmatrix}, \tag{2.8}$$

where D is a diagonal matrix whose entries are the inner products between the basis vectors in V and W.

Similarly to Arnoldi, the coefficients used to biorthogonalize V and W can be saved in an upper-Hessenberg matrix T as the bases are built, with the following relations being true [32]:

$$AV_k = V_k T_k + v_{k+1} t_{k+1,k} e_{k+1}^T \tag{2.9}$$

$$A^H W_k = W_k T_k^* + w_{k+1} t_{k+1,k}^* e_{k+1}^T \tag{2.10}$$

where t^* is the complex conjugate of t. As a consequence of (2.9) and (2.10), we have

$$W^H A V = DT \quad \text{and} \quad V^H A^H W = D^H T^* \implies DT = (T^*)^H D \tag{2.11}$$

A consequence of this is that the upper-Hessenberg matrix T is equal to a lower-Hessenberg matrix, meaning that it is actually tridiagonal. As a result the biorthogonalization of V and W can be implemented using a simple three term recurrence.

Practical implementations of Lanczos take advantage of this fact by using and storing only the tridiagonal entries of T. The ease of storing and factorizing tridiagonal matrices is the reason that some algorithms use Lanczos as the basis for solving linear systems of equations or eigenvalue problems, rather than Arnoldi.

2.2 Solving Linear Systems of Equations

Linear systems of equations are linear algebra problems of the form

$$Ax = b, \tag{2.12}$$

where A is an $n \times n$ matrix and x and b are n-dimensional vectors, and we seek to calculate the vector x that makes (2.12) true.

The most typical way to solve Equation (2.12) is to calculate a matrix factorization

$$A = BC \tag{2.13}$$

such that it is easy to solve linear systems of equations with both B and C as the left hand side matrices. The solution x can then be easily calculated as $x = C^{-1}B^{-1}b$, where the matrix inverse applications are performed implicitly through a linear system-solving procedure. Common matrix factorizations include LU factorizations (for general matrices A), in which $B = L$ is a lower triangular matrix and $C = U$ is an upper triangular matrix, or Cholesky factorizations (for symmetric positive definite A), in which $B = L$ is a lower triangular matrix and $C = L^H$. There are a variety of effective algorithms [30, 99] and software packages [1, 13, 78] for quickly and robustly calculating solutions this way, provided that matrix factorizations can be performed efficiently.

For the purposes of this dissertation, however, we will generally be operating under the constraints described in Section 2.1, in which A is too large to be factorized efficiently. Thus we need to solve Equation (2.12) using only matrix-vector multiplications with A, plus whatever dense matrix operations we care to use on matrices that are small enough for those operations to be feasible and efficient. Rather than finding the vector x that makes Equation (2.12) exactly true, we instead try to find a vector $\tilde{x}$ that makes it *approximately* true, in the sense that

$$\tilde{x} = \underset{\tilde{x}}{\operatorname{argmin}} ||b - A\tilde{x}||^2. \tag{2.14}$$

The vector $b - A\tilde{x}$ is called the linear system residual. We notably do not try to minimize the norm $||A^{-1}b - \tilde{x}||$, which at first blush would appear to be the proper quantity to minimize. This is purely due to practicality; because we do not know what the quantity $A^{-1}b$ is (that's what we are trying to calculate!), we can not compare prospective approximations $\tilde{x}$ against it. This has implications, as we will see later, for the behavior of linear system solving algorithms and for the behavior of eigenvalue algorithms that are based on them.

Since we are using only matrix-vector multiplications to do work with the matrix A, I assume that $\tilde{x} \in \mathcal{K}_k(A, b)$, where k is the maximum number of matrix-vector multiplications that we want to do[2] with A. If V is an $n \times (k+1)$ matrix whose column vectors are a basis for $\mathcal{K}_k(A, b)$, then $\tilde{x} = V\tilde{x}_v$ for some $(k+1)$ dimensional vector $\tilde{x}_v$, and the minimization problem that we have to solve in order to approximate the solution to Equation (2.12) becomes

$$\tilde{x}_v = \underset{\tilde{x}_v}{\operatorname{argmin}} ||b - AV\tilde{x}_v||^2. \tag{2.15}$$

Equation (2.15) is just a typical linear least squares problem, with the solution $\tilde{x}_v = (V^H A^H AV)^{-1} V^H A^H b$. The approximate solution $\tilde{x}$ for Equation (2.12) is then

$$\tilde{x} = V\tilde{x}_v = V(V^H A^H AV)^{-1} V^H A^H b \approx x \tag{2.16}$$

The original problem in Equation (2.12) is too difficult to solve directly by factorization methods due to the large dimension of A; on the other hand, the matrix is

[2]We may want to restrict the number of multiplications by A due to either memory constraints, time constraints, or both.

$(V^H A^H AV)$ is only $(k+1) \times (k+1)$, and so we can easily use factorization-based methods to calculate the application of its matrix inverse in Equation (2.16).

Equation (2.16) appears to be a neat and tidy solution for approximating the solution to Equation (2.12), but the challenge lies in the details of how V is calculated and, consequently, how the least squares problem in Equation (2.15) is solved. A naive first approach to calculating V might be to simply form the matrix

$$V = \begin{bmatrix} b & Ab & A^2b & \dots & A^kb \end{bmatrix}, \tag{2.17}$$

which can then be used in Equation (2.16) directly. This works very poorly in practice, however. The expression for V in Equation (2.17) would work just fine if it were used in a hypothetical computer with infinite numerical precision but, in real computers with finite precision, the column vectors of the right hand side of Equation (2.17) quickly become linearly dependent. The end result is that using Equation (2.16) to calculate $\tilde{x}$ produces an estimated solution of very poor accuracy; the residual norm $||b - A\tilde{x}||$ is much larger than it should be, and it does not decrease much when the dimension of the Krylov subspace is increased. Instead, practical algorithms use procedures based on Arnoldi or Lanczos iterations in order to apply expressions like Equation (2.16) in an efficient and numerically stable way.

The following subsections discuss several such practical algorithms for solving Equation (2.15). I focus primarily on algorithms that have clear and important connections to the FEAST algorithm, acknowledging that there are other algorithms in common use in addition the ones covered here. I refer the reader to other resources [73, 91, 102] for a look at other algorithms, as well as for the details of the efficient implementation of the algorithms that are discussed in the following subsections.

2.2.1 GMRES

GMRES [72] is a method for implementing Equation (2.16) in an efficient and numerically stable way. GMRES uses Arnoldi iterations to generate a basis V_k for $\mathcal{K}_k(A, b)$. The least squares problem that needs to be solved, Equation (2.15), then becomes

$$\tilde{x}_v = \underset{\tilde{x}_v}{\operatorname{argmin}} ||b - AV\tilde{x}_v||^2 = \underset{\tilde{x}_v}{\operatorname{argmin}} ||b_v - \tilde{H}_k\tilde{x}_v||^2, \tag{2.18}$$

which can be derived by using the Arnoldi relation $AV_k = V_{k+1}\tilde{H}_k$. The vector $b_v = V_{k+1}^H b$ is the projection of the right hand side b on to the Arnoldi basis V_{k+1}. Because the Arnoldi procedure uses $b/||b||$ as its its first basis vector, $b_v = ||b||e_1$ in practice, regardless of the value of k.

GMRES solves the least squares problem in Equation (2.18) by using the QR decomposition $\tilde{H}_k = QR$, i.e.

$$\tilde{x}_v = \underset{\tilde{x}_v}{\operatorname{argmin}} ||b_v - \tilde{H}_k\tilde{x}_v||^2 = \underset{\tilde{x}_v}{\operatorname{argmin}} ||b_v - QR\tilde{x}_v||^2 \tag{2.19}$$

$$= (R^H Q^H QR)^{-1} R^H Q^H b_v \tag{2.20}$$

$$= R^{-1} Q^H b_v. \tag{2.21}$$

The approximate linear system solution is then $\tilde{x} = V_k\tilde{x}_v$.

GMRES is typically implemented as an iterative procedure. Starting with a one-dimensional Krylov subspace basis V_1, each iteration of GMRES performs one iteration of Arnoldi in order to increase the dimension of V_k by one. The QR decomposition of $\tilde{H}_k$ is then updated, often using Householder transformations, and the matrix-vector product $Q^H b_v$ from Equation (2.21) is updated as well. The linear system residual norm is measured by using the $(k+1)^{\text{th}}$ entry of $Q^H b_v$, and if it is lower than some user-defined tolerance then then linear system $R\tilde{x}_v = Q^H b_v$ is solved (i.e. Equation (2.21) is evaluated) and $V_k\tilde{x}_v$ is returned as the approximate solution

to Equation (2.12). If the linear system residual norm is still too large, then another GMRES iteration is performed.

By iteratively building the Krylov subspace and checking the linear system residual norm, GMRES uses the minimum number of matrix-vector multiplications that is necessary to achieve the desired level of accuracy for an approximate solution. Iteratively updating the QR decomposition of $\tilde{H}_k$, and checking the linear system residual norm without actually solving the linear system $R\tilde{x}_v = Q^H b_v$ and calculating the norm $||b - AV\tilde{x}_v||$, are additional optimizations that improve the efficiency of the algorithm and minimize the amount of computation that needs to be done in order to arrive at an approximate solution.

GMRES is the fastest and most robust method for approximating the solutions to linear systems of equations, in the sense that it requires the lowest number of matrix-vector multiplications in order to arrive at an approximate solution $\tilde{x}$ with a linear system residual norm that is below a user-defined tolerance. This is because it uses the fewest shortcuts and approximations in calculating an approximate linear system solution; it is simply an efficient implementation of the mathematical procedure that minimizes the residual norm $||b - A\tilde{x}||$ for an approximate solution $\tilde{x} \in \mathcal{K}_k(A, b)$.

This speed and robustness comes at the price of high memory requirements. While GMRES can, in principle, use the lowest possible number of matrix-vector multiplications in order to approximate the solution to a linear system, it may be the case that this number of matrix-vector multiplications is larger than the number of Krylov subspace basis vectors that one can reasonably store on a computer. In that case GMRES must use restarts (see Section 2.2.5 on page 27), which will increase the required number of matrix-vector multiplications, and which may prevent GMRES from converging altogether.

Other algorithms for efficiently solving linear systems of equations seek to achieve better numerical performance in some sense by making approximations in the imple-

mentation of Equation (2.16), or even by using procedures that do not necessarily minimize the linear system residual norm for a given Krylov subspace. Algorithms like this may converge less robustly than GMRES, but with the benefit that they require substantially less memory, and can consequently use a Krylov subspace of much larger dimension. The following subsections describe a few such algorithms.

2.2.2 MINRES

Additional optimizations can be made to the GMRES algorithm when the matrix A is Hermitian, and the resulting algorithm is referred to as MINRES [59][3].

Hermitian matrices A are those for which $A = A^H$. The following is then true for a Krylov subspace basis V_k that is generated by Arnoldi:

$$H_k = V_k^H A V_k = V_k^H A^H V_k = H_k^H, \tag{2.22}$$

where H_k is the upper-Hessenberg matrix generated by the Arnoldi process. Because $H_k = H_k^H$, it must be a tridiagonal matrix. Similar to the Lanczos method, then, one only needs to store the tridiagonal elements of H_k, and the process of orthogonalizing V_k amounts to a simple, three-term recurrence.

MINRES takes advantage of the tridiagonality of H_k in order to avoid storing any matrices (such as H_k and V_k) at all. It stores only as many vectors as are required in order to, at each MINRES iteration, generate a new Krylov subspace basis vector that is orthogonal to all of the previous Krylov subspace vectors, update the Cholesky decomposition of the matrix $V^H A^H A V$ from Equation (2.16) by using an orthogonal decomposition (such as QR) of the matrix $\tilde{H}_k$, and then update the approximate linear system solution $\tilde{x}$. Because all of the matrices involved in this process - including the factorizations of the matrix $\tilde{H}_k$ - have only a small number of off-diagonal elements,

[3]Historically, it is worth noting, the order of events was actually reversed; MINRES was developed first, and GMRES was later developed as a generalized version of MINRES.

the procedure for updating them when the Krylov subspace dimension is increased by one can all be defined by short recurrence relations. The end result is that, at least in principle[4], MINRES can use a Krylov subspace of arbitrarily large dimension in order to approximate the solution to a linear system of equations, while using only a small, constant amount of computer memory. This can make it advantageous to use in place of GMRES when the performance of GMRES is substantially limited by the memory that is available for performing computations.

MINRES is not limited to being applied to only Hermitian matrices. It also works for *shifted* Hermitian matrices [21], i.e. matrices of the form $(zI - A)$, where $A^H = A$ and z is a complex scalar, because in this case $H_k = V_k^H(zI - A)V_k$ is also tridiagonal. This is useful when implementing the FEAST eigenvalue algorithm for standard Hermitian eigenvalue problems, because the primary computational task for FEAST in that case is to solve linear systems of the form $(zI - A)x = b$.

2.2.3 FOM

The Full Orthogonalization Method, or FOM, approximates the solution to $Ax = b$ as

$$\tilde{x} = V_k(V_k^H A V_k)^{-1} V_k^H b, \tag{2.23}$$

where V_k is the Krylov subspace basis that is generated by the Arnoldi process [73]. This is different, notably, from the approximation that is used by GMRES and MINRES, i.e. Equation (2.16). In general, then, FOM would be expected to require a larger Krylov subspace dimension, and therefore a larger number of matrix-vector multiplications, in order to achieve a given tolerance on the linear system residual norm $||b - A\tilde{x}||$, because it calculates an approximate solution $\tilde{x}$ by using an expression that does not minimize the residual norm for a given Krylov subspace.

---

[4]In practice, things are often less ideal; finite numerical precision and less-than-ideal conditioning of the matrix A can cause MINRES to perform less well than one might expect it to based on the dimension of the Krylov subspace that it uses [59].

An exception to this is when the matrix A is symmetric and positive-definite (SPD). In that case the matrix A^{-1} defines an inner product, and

$$\tilde{x}_v = (V_k^H A V_k)^{-1} V_k^H b \tag{2.24}$$

is the solution to the problem [59]

$$\tilde{x}_v = \underset{\tilde{x}_v}{\operatorname{argmin}} \, ||b - A V_k x_v||_{A^{-1}}^2, \tag{2.25}$$

where

$$||b - A V_k x_v||_{A^{-1}}^2 = (b - A V_k x_v)^H A^{-1} (b - A V_k x_v) \tag{2.26}$$

is the square of the norm of the linear system residual $b - AV_k x_v$, using the inner product defined by A^{-1}.

When A is symmetric and positive definite then $H_k = V^H A V$ is again tridiagonal and, as with MINRES, one can implement FOM by using short recurrence relations, thereby avoiding the storage of the matrices V_k, H_k, and the Cholesky decomposition of H_k. FOM implemented with short recurrence relations is the Conjugate Gradients algorithm (CG) [73].

Conjugate Gradients is a historically important algorithm, and it is highly effective for the problem domains where it is primarily used (i.e. problems with SPD matrices). It will not be discussed any further in this dissertation, however, since it has no practical implications for the FEAST algorithm; the linear systems of equations that must be solved in the FEAST algorithm are almost never symmetric positive definite, and so Conjugate Gradients is not an appropriate algorithm to use for their solution.

FOM itself does not necessarily have much to recommend it as an algorithm. Its primary benefit is its simplicity; apart from that, for any problem domain where one might consider applying FOM, another algorithm (such as GMRES or Conjugate

Gradients) would almost certainly be more effective. I mention it here, however, because it will come up later in explaining the relationship between the FEAST algorithm and certain other methods for solving eigenvalue problems.

2.2.4 BiCG

The Bicongjugate Gradients algorithm [16, 48], or BiCG, approximates the solution to two linear systems of equations $Ax = b$ and $A^H y = c$ as [73]

$$\tilde{x} = V(W_k^H A V_k)^{-1} W_k^H b \tag{2.27}$$

$$\tilde{y} = W(V_k^H A^H W_k)^{-1} V_k^H c, \tag{2.28}$$

where V_k and W_k are matrices of basis vectors generated by Lanczos for the Krylov subspaces $\mathcal{K}_k(A, b)$ and $\mathcal{K}_k(A^H, c)$, respectively. For most applications one only needs to solve the first linear system, $Ax = b$, and so the basis W_k is generated for the Krylov subspace $\mathcal{K}_k(A^H, b)$ instead, without being used to solve a second linear system of equations with A^H.

Unlike the Arnoldi process, which only generates a tridiagonal matrix $H_k = V^H A V$ when A belongs to certain restricted classes of matrices (such as shifted Hermitian matrices), Lanczos Biorthogonalization produces a tridiagonal matrix $T_k = W_k^H A V_k$ for *any* matrix A. BiCG is thus implemented by using short recurrence relations, with the matrices V_k, W_k, T_k, and factorizations of T_k never actually being stored. Only a small number of storage vectors are used in order to iteratively update the approximate solution $\tilde{x}$ and build (implicitly) the subspaces V_k and W_k.

BiCG has a few qualities that prevent it from being a first-choice algorithm for solving linear systems of equations. One of these is that it does not minimize the residual norm $||b - AV_k\tilde{x}_v||$, and so it can be expected to take longer to converge than would an algorithm like GMRES when the tolerance for convergence is based on the

residual norm. In practice, moreover, its convergence tends to be erratic, rather than always reducing the residual norm.

BiCG also requires matrix-vector products with the matrix A^H. Most applications require only the solution of the linear system $Ax = b$, not the system $A^H y = c$; the matrix-vector products with A^H are thus "wasted", in the sense that they are only used to calculate coefficients for recurrences rather than being used to solve linear systems. There are even some applications in which matrix-vector products with A^H can not actually be computed due to the way that the matrix A is stored, making BiCG impossible to implement altogether.

BiCG can also suffer from "breakdowns", in which the recurrence relations that are used to build the matrix T_k or its LU decomposition fail [32]. There are methods available that can prevent such breakdowns, at the price of increasing the complexity of the algorithm's implementation.

Similarly to FOM, I mention BiCG not because it is an algorithm of choice for solving linear systems of equations, but because it will later serve as a connection between the FEAST algorithm and other methods of solving eigenvalue problems. There are better algorithms available that use a short recurrence relation with non-symmetric matrices, such as BiCGSTAB [86, 101], which replaces the steps in BiCG involving matrix-vector products with A^H with a recurrence involving multiplication by A instead. One of these steps involves minimizing the norm of the residual with respect to multiplication by a one degree polynomial of A (i.e. a single iteration of GMRES), which helps to stabilize the convergence of the residual norm.

2.2.5 Restarts

Sometimes one begins solving a linear system of equations

$$Ax = b \tag{2.29}$$

with a reasonable initial guess $\tilde{x}$ for its approximate solution. Other times, particularly when using an algorithm like GMRES whose performance is limited by the available memory on a computer, the approximate solution to a linear system of equations may not be able converge to have a residual norm that is below a user-defined tolerance. In situations like these the process of solving the linear system can be "restarted". The approximate solution $\tilde{x}$ and the linear system residual $r = b - A\tilde{x}$ can be used to form a new linear system of equations, such that the approximate solution to this new set of equations gives an update that will improve on the original approximate solution $\tilde{x}$.

The way that this is usually done is to assume that the exact solution x for Equation (2.29) takes the form $x = \tilde{x} + \delta x$, where $\tilde{x}$ is some approximate solution with residual $r_L = b - A\tilde{x}$, and δx is a correction to the approximate solution that gives an exact solution. The equation

$$A\tilde{x} = b - r_L \tag{2.30}$$

is true by definition, and exactly solving the new linear system of equations

$$A\delta x = r_L \tag{2.31}$$

will produce a δx that corrects $\tilde{x}$ to give the exact solution. This can be seen by adding Equations (2.30) and (2.31) together:

$$A\tilde{x} + A\delta x = b - r_L + r_L \tag{2.32}$$

$$A(\tilde{x} + \delta x) = b \tag{2.33}$$

This process is also known as *iterative refinement* [106].

In the context of restarting the solution of linear systems of equations, Equation (2.31) is, itself, solved approximately by using the same algorithm that was used to find an initial estimated solution $\tilde{x}$ to Equation (2.29). The approximate solution $\delta\tilde{x}$ for Equation (2.31) is then used to update $\tilde{x}$ as

$$\tilde{x} \leftarrow \tilde{x} + \delta\tilde{x}. \tag{2.34}$$

If the residual for the initial approximate solution of Equation (2.31) is $\delta r_L = r_L - A\delta\tilde{x}$, then the residual for the new approximate solution $\tilde{x} + \tilde{\delta}x$ is

$$r_{new} = b - A(\tilde{x} + \delta\tilde{x}) \tag{2.35}$$

$$= (b - A\tilde{x}) - A\delta\tilde{x} \tag{2.36}$$

$$= r_L - r_L + \delta r_L = \delta r_L. \tag{2.37}$$

The total residual thus becomes the residual for the correction equation (2.31). As long as the relative residual norm for the solution of Equation (2.31) is less than 1, i.e.

$$\frac{||\delta r_L||}{||r_L||} < 1, \tag{2.38}$$

the new, corrected solution $\tilde{x} + \tilde{x}_\delta$ is guaranteed to be more accurate than the initial solution $\tilde{x}$.

The process of re-forming and re-solving Equation (2.31) can thus be repeated until the norm $||r_L|| = ||b - A\tilde{x}||$ is sufficiently low. This is what is usually referred to as restarting.

This restarting procedure is so commonly-used that many sources will actually define the linear system solving algorithms that are discussed in this chapter in terms of solving Equation (2.31) given some initial guess $\tilde{x} = x_0$ for the solution to Equation (2.29), rather than in terms of solving Equation (2.29), which is considered to be the

special case of Equation (2.31) when $x_0 = 0$. For the purposes of this dissertation, however, it is worth noting that Equation (2.31) is not a unique choice for calculating an update to an approximate solution for a linear system of equations.

Equation (2.31) is the update equation that is produced by assuming that the relationship between the exact solution, the approximate solution, and the update vector is $x = \tilde{x} + \delta x$, i.e. Equation (2.33). The relationship between these three quantities can actually be anything that we want it to be, though; different choices for how to relate x, $\tilde{x}$, and δx simply produce different equations that need to be solved in order to calculate δx. We will see later on that the choice

$$x = \frac{(\tilde{x} + \delta x)}{z - \tilde{\lambda}}, \tag{2.39}$$

where z and $\tilde{\lambda}$ are some particular complex scalars, can be very useful in the context of implementing the FEAST eigenvalue algorithm using approximate linear system solving algorithms of the kind that are discussed in this chapter. Inserting Equation (2.39) into Equation (2.29) produces a linear system correction equation of the form

$$A\delta x = (z - \tilde{\lambda})b - A\tilde{x}, \tag{2.40}$$

where the right hand side is no longer equal to the linear system residual $r_L = b - A\tilde{x}$. This form for the correction equation turns out to be more useful than Equation (2.31), because the FEAST algorithm naturally provides initial guesses for the solution to the linear system in Equation (2.29) that are related to eigenvalue estimates $\tilde{\lambda}$ and complex shifts z. Incorporating that information into the solution of the FEAST linear systems of equations can have substantial performance benefits.

2.3 Solving Eigenvalue Problems

Generalized eigenvalue problems consist of identifying vectors x_R and x_L, and scalars λ, that satisfy

$$Ax_R = \lambda B x_R \tag{2.41}$$

$$A^H x_L = \lambda^* B^H x_L, \tag{2.42}$$

where A and B are $n \times n$ matrices. The pair (A, B) is referred to as a "matrix pencil". The vectors x_R are called "right eigenvectors" (because they are eigenvectors when multiplied on the right of A), the vectors x_L are called "left eigenvectors" (because they are eigenvectors when multiplied on the left of A, i.e. $x_L^H A = x_L^H B\lambda$), and the scalars λ are the eigenvalues. Non-defective matrices[5] have n eigentriples of x_R, x_L, and λ such that Equations (2.41) and (2.41) are true, and solving an eigenvalue problem consists of identifying some or all such eigentriples.

If the right and left eigenvectors x_R and x_L are collected as column vectors into matrices X_R and X_L such that

$$X_R = \left[x_{R1} \;\; x_{R2} \;\; \ldots \;\; x_{Rn}\right], \quad X_L = \left[x_{L1} \;\; x_{L2} \;\; \ldots \;\; x_{Ln}\right], \tag{2.43}$$

and the eigenvalues λ are collected into a diagonal matrix Λ such that

$$\Lambda = \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{bmatrix}, \tag{2.44}$$

[5] I only consider non-defective matrices in this dissertation, and I will hereafter assume that all matrices under consideration are non-defective.

then

$$A = BX_R \Lambda X_L^H B \tag{2.45}$$

and

$$X_L^H B = X_R^{-1}. \tag{2.46}$$

Equation (2.45) is the eigenvalue decomposition of A for the matrix pencil (A, B), and Equation (2.46) implies the B-biorthogonality of X_R and X_L, i.e. $X_L^H B X_R = I$ and $X_R^H B^H X_L = I$.

When the matrix A is Hermitian, i.e. $A = A^H$, and B is symmetric and positive definite, then $X_R = X_L$ and all of the eigenvalues are real numbers. When $B = I$ then Equations (2.41) and (2.42) become

$$Ax_R = \lambda x_R \tag{2.47}$$

$$A^H x_L = \lambda^* x_L, \tag{2.48}$$

which is called the "standard eigenvalue problem". In this case $X_L^H = X_R^{-1}$, i.e. the column vectors of X_L and X_R are biorthogonal.

The most common goal in solving eigenvalue problems is to calculate the right eigenvectors for a standard eigenvalue problem. Standard eigenvalue problems are more common and easier to solve than are generalized eigenvalue problems, and the left eigenvectors are less likely to be useful than the right eigenvectors in practical applications. Any method for solving standard eigenvalue problems can, at least in principle, also by applied to solving generalized eigenvalue problems by noting that, for example, multiplying Equation (2.41) on the left by B^{-1} turns the generalized eigenvalue problem into the standard one:

$$Ax = \lambda Bx \longrightarrow B^{-1}Ax = \lambda x \tag{2.49}$$

This is not always a good idea, though; it is not necessarily practical or efficient to form the matrix $B^{-1}A$. Even with algorithms that use only matrix-vector multiplication for solving eigenvalue problems, every multiplication by A must then be followed by a linear system solution with B, which can make the eigenvalue problem more expensive to solve. For these reasons it can be advantageous to solve generalized eigenvalue problems without first converting them into standard eigenvalue problems.

Similarly, any algorithm that can be used to calculate the right eigenvectors of an eigenvalue problem can also be used to calculate the left eigenvectors, provided that the matrices A^H and B^H are available. It is sometimes advantageous to calculate both sets of eigenvectors simultaneously, and this is true of the FEAST algorithm when solving non-symmetric eigenvalue problems.

As with linear systems of equations, there are well-known factorization-based algorithms for solving both generalized and standard eigenvalue problems, such as the QZ and QR algorithms [30, 99]. These algorithms calculate *all* of the n eigenvalues and eigenvectors for a given $n \times n$ matrix pencil. Also as with linear systems of equations, I will consider "small" eigenvalue problems to be those that can be efficiently or easily solved with these factorization-based algorithms.

I will consider "large" eigenvalue problems to be those for which it is impossible to calculate all n eigenvalues and eigenvectors. With large matrix pencils, the matrices are stored in some way that makes matrix-vector multiplications efficient to calculate, but which avoids storing all of the coordinates of the matrices. The matrices A and B may be stored in a sparse format, wherein most of the coordinates are zero and only the nonzero coordinates are actually stored explicitly. Or, they may be stored implicitly in the form of computer routines that calculate the result of matrix-vector multiplication when given an input vector; see references [27] and [28] for examples of quantum simulation projects that use this storage method. In this second case, it is worth noting, it is sometimes not possible to calculate matrix-vector multiplications

with A^H or B^H, which necessarily constrains the algorithms that can be applied to solving such an eigenvalue problem.

There are also "medium" sized eigenvalue problems. These are eigenvalue problems whose matrix pencils are small enough that there exist efficient factorization-based methods for solving linear systems of equations with them, but which are large enough that calculating all n eigenvalues and eigenvectors is very difficult or inefficient. Eigenvalue problems like this can be solved particularly quickly by taking advantage of efficient factorization-based linear system solving algorithms.

The goal with either medium-sized or large-sized eigenvalue problems is to calculate only a small number $m \ll n$ of eigenvalues and eigenvectors, usually specific ones that are important for a given application. The following subsections describe several algorithms for solving eigenvalue problems in this way, with an emphasis on methods of solving eigenvalue problems that are related to the FEAST algorithm.

2.3.1 Rayleigh-Ritz

The feature that most of the eigenvalue algorithms in this chapter share is that they solve large eigenvalue problems by projecting them on to a smaller subspace in order to form a small eigenvalue problem. The resulting small eigenvalue problem can then be solved easily by using standard algorithms and software packages for efficiently calculating the eigenvalue decompositions of dense matrices and matrix pencils.

If we want to approximate the right eigenvectors, $\tilde{x}$, of an n-dimensional matrix pencil (A, B) using an m-dimensional subspace spanned by the matrix of basis vectors V, with the condition that the residuals $\tilde{\lambda}B\tilde{x} - A\tilde{x}$ be orthogonal to the m-dimensional subspace spanned by V, then the eigenvalue problem that we need to solve becomes [71]

$$V^H A V \tilde{x}_v = \tilde{\lambda} V^H B V x_v \tag{2.50}$$

The eigenvalues $\tilde{\lambda}$ are then approximations of m eigenvalues of (A, B), and the vectors $\tilde{x} = Vx_v$ are approximations of m eigenvectors of (A, B). Whereas the original matrix pencil (A, B) was of dimension n, and was potentially quite expensive to solve, the new matrix pencil $(V^H AV, V^H BV)$ is of dimension $m \ll n$, and is therefore more easily solved by using standard methods. This is the Rayleigh-Ritz method, with the approximate eigenvalues $\tilde{\lambda}$ often being referred to as "Ritz values", and the approximate eigenvectors Vx_v often being referred to as "Ritz vectors".

When V is a basis for a Krylov subspace, the standard Rayleigh-Ritz method is most effective for finding approximations of the eigenvectors whose eigenvalues are located in the exterior of the spectrum [55, 56]. For Hermitian matrix pencils, whose eigenvalues are all real numbers, this means that it tends to approximate the most-positive and the most-negative eigenvalues. For nonsymmetric matrix pencils, whose eigenvalues can be anywhere in the complex plane, this means that it tends to approximate the eigenvalues that are close to the edge of the convex hull of the set of eigenvalues of (A, B).

The interior eigenvalues can be more effectively approximated by using a different projected eigenvalue problem:

$$V^H(zB - A)^H(zB - A)Vx_v = (z - \lambda)V^H(zB - A)^H Vx_v. \tag{2.51}$$

Approximating the eigenvalues and eigenvectors of (A, B) using the solution to Equation (2.51) is called the Harmonic Ritz method [55,56,91]. It produces approximations of the eigenvectors whose eigenvalues are closest to the complex scalar z in the complex plane.

It is also possible to use oblique projection in order to produce a reduced-dimension eigenvalue problem, rather than using the orthogonal projection in Equation (2.50). If we constrain the right-eigenvector residuals $\tilde{\lambda}B\tilde{x} - A\tilde{x}$ to be orthogonal to the

subspace spanned by the basis vectors W (rather than the one spanned by V as in Equation (2.50)), then the resulting reduced-dimension eigenvalue problem is

$$W^H A V x_v = \tilde{\lambda} W^H B V x_v. \tag{2.52}$$

One the benefits of oblique projection is that it allows the simultaneous estimation of both the left and right eigenvectors for nonsymmetric matrix pencils. The right eigenvectors of (A, B) are approximated by Vx_v as in standard Rayleigh-Ritz, and the left eigenvectors are approximated by Wx_w, where the vectors x_w are the left eigenvectors for the matrix pencil $(W^H AV, W^H BV)$. Oblique projection also makes it possible to take advantage of algorithms that generate biorthogonal Krylov basis sets, such as Lanczos biorthogonalization.

2.3.2 Arnoldi and Lanczos

As discussed in Section 2.1.1, Arnoldi iterations can be used to produce an orthonormal basis set V_k for the $(k+1)$-dimensional Krylov subspace $\mathcal{K}_k(A, x_0)$. The output of Arnoldi iterations are two matrices H_k and V_k such that

$$V_k^H A V_k = H_k, \quad V_k^H V_k = I_{k+1,k+1}. \tag{2.53}$$

Arnoldi thus naturally produces the projected eigenvalue problem that is used by the Rayleigh-Ritz method for approximating the right eigenvectors of the standard eigenvalue problem $Ax = \lambda x$ in the subspace spanned by V_k, i.e.

$$H_k x_v = \tilde{\lambda} x_v. \tag{2.54}$$

The solutions to the $(k+1)$-dimensional standard eigenvalue problem in Equation (2.54) are approximations of the eigenvalues and eigenvectors of the matrix pencil

(A, I), with $\tilde{\lambda}$ being the Ritz values for the subspace basis V_k and the vectors $V_k x_v$ being the Ritz vectors.

Using Arnoldi iterations in this way has a nice interpretation in terms of matrix polynomials. Using Arnoldi iterations with Rayleigh-Ritz is equivalent to solving the minimization problem [99]

$$p_{k+1} = \min_{p_{k+1}} ||p_{k+1}(A)x_0||, \tag{2.55}$$

where p_{k+1} is a monic polynomial of degree $k+1$. The solution to (2.55) is

$$p_{k+1}(z) = \prod_{i=1}^{k+1}(z - \tilde{\lambda}_i) \tag{2.56}$$

Arnoldi thus approximates the characteristic polynomial of the matrix A with a degree-$(k+1)$ polynomial, by finding the degree-$(k+1)$ polynomial that minimizes the matrix polynomial product with an initial guess vector x_0. The roots of this degree-$(k+1)$ polynomial are the Ritz values of H_k.

Lanczos biorthogonalization, which produces biorthogonal sets of basis vectors W_k and V_k for the subspaces $\mathcal{K}_k(A^H, x_0)$ and $\mathcal{K}_k(A, x_0)$ (respectively), can be used for solving eigenvalue problems in essentially the same way as Arnoldi. Lanczos outputs the matrices W_k, V_k, and T_k such that

$$W_k^H A V_k = T_k, \quad W_k^H V_k = I_{k+1,k+1}. \tag{2.57}$$

Similarly to Arnoldi, then, Lanczos naturally generates the reduced-dimension eigenvalue problem that is produced when using a Rayleigh-Ritz-like oblique projection to estimate both the left and the right eigenvectors of the matrix pencil (A, I), i.e.

$$\begin{aligned} T_k x_v &= \tilde{\lambda} x_v \\ T_k^H x_w &= \tilde{\lambda}^* x_w \end{aligned} \tag{2.58}$$

The vectors Wx_w and Vx_v are Ritz vectors that approximate the left and right eigenvectors of (A, I), and the scalars $\tilde{\lambda}$ are the corresponding Ritz values.

Beyond estimating both the left and right eigenvectors simultaneously, Lanczos has the benefit that the matrix T_k is tridiagonal, which requires less computer memory storage and which makes the solution of the eigenvalue problem in Equation (2.58) easier. Unlike with Arnoldi, Lanczos can even be used to calculate eigenvalue and eigenvector estimates without storing the basis vectors W_k and V_k by using three term recurrences, at the cost of having to repeat the Lanczos process twice in order to reconstruct the desired eigenvectors.

Arnoldi and Lanczos both operate most naturally on standard eigenvalue problems $Ax = \lambda x$. They *can* be used to solve generalized eigenvalue problems $Ax = \lambda Bx$ by[6] multiplying each side by B^{-1}, i.e. by solving the standard eigenvalue problem

$$B^{-1}Ax = \lambda x, \tag{2.59}$$

but this strategy is limited by the dimension of the eigenvalue problem. This is because solving Equation (2.59) with Lanczos or Arnoldi requires forming basis sets for Krylov subspaces like $\mathcal{K}_k(B^{-1}A, x_0)$; every matrix multiplication by A when forming such a basis set must be accompanied by the solution of a linear system of equations with the matrix B. For "medium" sized eigenvalue problems, in which the matrices A and B are small enough that they can be efficiently factorized for solving linear systems of equations, this can be a reasonably efficient process. For large eigenvalue problems, where A and B are large enough that the exact solution of linear systems using matrix factorizations is infeasible, using Equation (2.59) with Arnoldi or Lanczos will likely be prohibitively expensive.

[6]This is not the only method of converting generalized eigenvalue problems into standard ones, but the challenges remain the same regardless of which method is used.

2.3.3 Krylov Restarts

It is often the case that Arnoldi or Lanczos will fail to converge to a sufficiently accurate approximation of the desired eigenvalues and eigenvectors before exceeding some limit on the available computer memory[7]. When this happens, much as with solving linear systems of equations, the Krylov iterations can be restarted.

Restarting the solution of an eigenvalue problem consists of simply redoing Arnoldi or Lanczos iterations using a new initial guess. If the original Krylov subspace that was used for approximating eigenvectors was $\mathcal{K}_k(A, \tilde{x}^{(0)})$, with corresponding matrix of basis vectors V_k, then restarting means building a new set of basis vectors for the subspace $\mathcal{K}_k(A, \tilde{x}^{(1)})$, with

$$\tilde{x}^{(1)} = Vc = p_{k+1}(A)\tilde{x}^{(0)}. \tag{2.60}$$

The $(k+1)$-dimensional vector c is a vector of coefficients that is used to select an element of the subspace $\mathcal{K}_k(A, \tilde{x}^{(0)})$ for use as a new initial guess. Because $\mathcal{K}_k(A, \tilde{x}^{(0)})$ is just the subspace that spans all vectors that can result from the application of a degree k matrix polynomial of A to the first initial guess, $\tilde{x}^{(0)}$, the act of choosing a particular vector $\tilde{x}^{(1)} = Vc$ to restart with is equivalent to developing a new initial guess by applying a polynomial filter of A to $\tilde{x}^{(0)}$.

There are a variety of different ways that one can choose a vector $\tilde{x}^{(1)}$ with which to restart [54,70,88]. The simplest is to choose $\tilde{x}^{(1)}$ to be the Ritz vector whose Ritz value is closest to an eigenvalue that one is interested in calculating. If, for example, we are interested in using Arnoldi to calculate the eigenvector for the lowest eigenvalue of the pencil (A, I), then we would select $\tilde{x}^{(1)}$ to be the Arnoldi Ritz vector corresponding to the lowest Arnoldi Ritz value. This is not necessarily the best method for choosing

[7]Such a limit might be imposed by the actual amount of computer memory that is available, or it might be imposed by the user in order to avoid increasingly expensive orthogonalizations and eigenvalue problem solutions as the Krylov subspace dimension grows (in the case of Arnoldi).

vectors with which to restart Krylov methods, but it will be important in explaining the relationship between the FEAST algorithm and Arnoldi.

2.3.4 Subspace Iteration

Subspace Iteration is a method for calculating m eigenvalues and eigenvectors using a subspace of constant dimension m (unlike Arnoldi or Lanczos, for example, which increase their subspace dimension by 1 at each iteration, unless one chooses to restart). The procedure for Subspace Iteration is described in Algorithm 1. With

Algorithm 1 Subspace Iterations

Inputs:

- $n \times n$ matrix A
- $n \times m$ matrix $\tilde{X}^{(0)}$ whose column vectors are (possibly random) initial guesses for the dominant eigenvectors of the pencil (A, I).
- Stopping tolerance ϵ

For each iteration i:

1. Calculate $V = A\tilde{X}^{(i)}$
2. Orthonormalize V
3. Solve $m \times m$ eigenvalue problem $V^H AVX_v = X_v\tilde{\Lambda}$ for $m \times m$ matrix of eigenvectors X_v and diagonal matrix of eigenvalues $\tilde{\Lambda}$
4. Set $\tilde{X}^{(i+1)} = VX_v$
5. Calculate block of residual vectors $R = B\tilde{X}^{(i+1)}\tilde{\Lambda} - A\tilde{X}^{(i+1)}$. If the norms of all of the column vectors of R are less than ϵ, **stop**. Otherwise **goto** Step 1.

Outputs: diagonal matrix $\tilde{\Lambda}$ of approximations for the m largest-magnitude eigenvalues, and approximations for the corresponding eigenvectors $\tilde{X}$.

Subspace Iteration, a block $\tilde{X}$ of m estimated eigenvectors is refined by performing matrix-vector multiplications with the matrix A. The refined, approximate eigenvectors are then used as a subspace in which to perform the Rayleigh-Ritz procedure, which returns new estimates for eigenvectors and eigenvalues. If the new estimates are sufficiently accurate then the estimated eigenvalues and eigenvectors are returned; otherwise the same procedure is repeated again.

Subspace Iteration converges to the eigenvectors corresponding to the m largest-magnitude eigenvalues of the matrix pencil (A, I). This is the sense in which matrix multiplication by A acts as a "refinement" of the approximate eigenvector subspace: if an approximate eigenvector $\tilde{x}$ is expressed as a linear combination of the exact eigenvectors x_i, then multiplication by A amounts to scaling each component of $\tilde{x}$ in the x_i basis by its corresponding eigenvalue λ_i, i.e.

$$A\tilde{x} = \sum_{i=1}^{n} c_i A x_i = \sum_{i=1}^{n} c_i \lambda_i x_i. \tag{2.61}$$

Intuitively, the magnitudes of the components of $A\tilde{x}$ grow the most in the direction of the eigenvectors whose eigenvalues have the largest magnitude. Repeated multiplication of a single vector by A, interspersed with normalization, eventually produces a vector that is equal to the eigenvector with the largest magnitude eigenvalue. Repeated multiplication of a collection of m vectors $\tilde{X}$ by A, interspersed with orthonormalization of that collection of vectors, produces a basis set that spans the m eigenvectors corresponding to the m largest-magnitude eigenvalues of A.

This notion of convergence can be written quantitatively in a simple way. If q_j is an estimate for the eigenvector x_j in the subspace spanned by $\tilde{X}$ at the start of a single subspace iteration, and $\tilde{q}_j$ is an estimate for x_j in the subspace spanned by $\tilde{X}$ at the end of a single subspace iteration, then the following inequality holds true [71,74]:

$$||\tilde{q}_j - x_j|| \leq \left(\frac{|\lambda_{m+1}|}{|\lambda_j|}\right) ||q_j - x_j||, \tag{2.62}$$

where the eigenvalues of A are ordered such that $|\lambda_j| > |\lambda_{j+1}|$. A single Subspace Iteration, using a subspace of dimension m, thus reduces the error for the estimation of eigenvector j in proportion to the ratio between the $(m+1)^{\text{th}}$ largest eigenvalue and the j^{th} largest eigenvalue; Subspace Iteration is therefore said to converge linearly. The details for the derivation of Equation (2.62) are provided in the appendix in Section A.1.

Equation (2.62) tells us that Subspace Iteration converges to an accurate estimate for the j^{th} eigenvector more quickly if the $(m+1)^{\text{th}}$ largest eigenvalue of A has a much smaller magnitude than the j^{th} eigenvalue of A. For this reason Subspace Iteration is more efficient for some matrices than for others, depending on the eigenvalue spectrum, and one can increase the rate of convergence to the j^{th} eigenvector by using a larger subspace dimension m.

Similarly to Arnoldi and Lanczos, Subspace Iteration operates naturally on matrix pencils of the form (A, I), and generalized eigenvalue problems for (A, B) can be solved by using Subspace Iteration with $(B^{-1}A, I)$. This relies, again, on being able to efficiently factorize B so that linear systems of equations can be solved with it. For large, generalized eigenvalue problems, where any kind of matrix factorization is assumed to be prohibitively difficult, Subspace Iteration is not an efficient method of solution.

Subspace Iteration is also a naturally blocked algorithm; it operates on blocks of m vectors at a time. It is closely related to block implementations of Arnoldi. Subspace Iteration is the same as performing block Arnoldi iterations with a Krylov subspace of order $k = 0$, using restarts such that the restart polynomial filter (see Equation (2.60)) is always $p(A) = A$. Standard Subspace Iteration can therefore be seen as a much less effective implementation of restarted block Arnoldi, in that it does not use most of the information that can be gained by building and using a Krylov subspace. It does, however, have the benefit over block Arnoldi of requiring a constant subspace dimension of only m, whereas the dimension of a block Arnoldi subspace is $(k+1)m$ for k iterations of Arnoldi.

The properties of Subspace Iteration that were described in this section will be revisited again when discussing the FEAST algorithm in Chapter 3, because the FEAST algorithm is perhaps best-interpreted as a more sophisticated version of subspace iterations.

2.3.5 Shift and Invert Iterations

Standard Subspace Iteration usually compares unfavorably with Krylov methods like Arnoldi and Lanczos because of its limitations in terms of both convergence rate and the eigenvectors that it can approximate. Subspace Iteration tends to converge relatively slowly (depending on the spectrum of the eigenvalue problem), and it can only find the largest-magnitude eigenvalues and corresponding eigenvectors.

Shift and Invert Iteration, which is summarized in Algorithm 2, is a modified version of Subspace Iteration that uses the solution of linear systems of equations in order to address both of these shortcomings. The solution of linear systems allows Shift and Invert Iteration to calculate eigenvectors whose eigenvalues are located near a particular, user-defined scalar z in the complex plane, and to do so almost arbitrarily quickly (in a sense that will be made more concrete shortly).

Algorithm 2 Shift and Invert Iteration

Inputs:

- $n \times n$ matrix A
- $n \times m$ matrix $\tilde{X}^{(0)}$ whose column vectors are (possibly random) initial guesses for the eigenvectors of the pencil (A, I) whose eigenvalues are closest to some complex scalar z.
- Stopping tolerance ϵ
- Complex scalar shift z located near the eigenvalues that are to be calculated.

For each iteration i:

1. Calculate $V = (zI - A)^{-1}\tilde{X}^{(i)}$
2. Orthonormalize V
3. Solve $m \times m$ eigenvalue problem $V^H AVX_v = X_v\tilde{\Lambda}$ for $m \times m$ matrix of eigenvectors X_v and diagonal matrix of eigenvalues $\tilde{\Lambda}$
4. Set $\tilde{X}^{(i+1)} = VX_v$
5. Calculate block of residual vectors $R = \tilde{X}^{(i+1)}\tilde{\Lambda} - A\tilde{X}^{(i+1)}$. If the norms of all of the column vectors of R are less than ϵ, **stop**. Otherwise **goto** Step 1.

Outputs: diagonal matrix $\tilde{\Lambda}$ of approximations for the m eigenvalues closest to z, and approximations for the corresponding eigenvectors $\tilde{X}$.

Shift and Invert Iteration is exactly the same as Subspace Iteration, except for Step 1: in Step 1, multiplication by A in Subspace Iteration is replaced with multiplication by $(zI-A)^{-1}$ in Shift and Invert Iteration. The multiplication $(zI-A)^{-1}\tilde{X}^{(i)}$ in Step 1 is performed by solving m linear systems of equations (one for each column vector of $X^{(i)}$) using whatever factorization-based method one prefers.

The theory for the convergence of Shift and Invert Iteration (i.e. Equation (2.62) is the same as the theory for the convergence of Subspace Iteration, with the eigenvalues of A being replaced by the eigenvalues of $(zI-A)^{-1}$. The Shift and Invert version of Equation (2.62) is then

$$||\tilde{q}_j - x_j|| \leq \left(\frac{|z-\lambda_j|}{|z-\lambda_{m+1}|}\right) ||q_j - x_j||, \tag{2.63}$$

where the eigenvalues λ_j of A are now numbered such that $|z-\lambda_j| < |z-\lambda_{j+1}|$. Note that λ_j is now in the *numerator* in Equation (2.63), because the eigenvalues of $(zI-A)^{-1}$ are $1/(z-\lambda_j)$ and

$$\frac{|1/(z-\lambda_{m+1})|}{|1/(z-\lambda_j)|} = \frac{|z-\lambda_j|}{|z-\lambda_{m+1}|} \tag{2.64}$$

Shift and Invert Iteration therefore converges to the m eigenvectors whose eigenvalues that are closest to the shift z. The rate at which they converge, moreover, is determined by the location the shift z; the closer z is to a particular eigenvalue λ_j, the faster Shift and Invert Iteration will converge to a good approximation of the corresponding eigenvector x_j. This is somewhat intuitive, in that the operation $f(A) = (zI-A)^{-1}$ maps the matrix A to a new matrix $(zI-A)^{-1}$ that has the same eigenvectors as A, with corresponding eigenvalues such that the eigenvalues λ_j of A are mapped to eigenvalues $1/(z-\lambda_j)$ of $(zI-A)^{-1}$. The eigenvalues $1/(z-\lambda_j)$ will have a very large magnitude for values of λ_j that are close to z, and so Sub-

space Iteration with the matrix $(zI - A)^{-1}$ will converge quickly to the corresponding eigenvectors.

A potential downside of Shift and Invert Iteration is that different eigenvalues will converge at potentially very different rates. The eigenvalue closest to the shift z will converge very quickly, but the eigenvalue that is *second*-closest to z will likely converge much more slowly than that, and with the third-closest converging even more slowly, and so on. Shift and Invert Iteration tends to be most efficient for calculating small groups of eigenvalues that are all clustered close together.

Shift and Invert Iteration, as described so far, is only appropriate for the "medium" sized eigenvalue problems where factorization-based algorithms for calculating eigenvalue decompositions are too expensive, but factorization-based algorithms for solving linear systems of equations are not. As a result, Shift and Invert Iteration is often used for generalized eigenvalue problems as well, which always require a linear system solution of some kind. Generalized eigenvalue problems for matrix pencils (A, B) can be efficiently solved by applying Shift and Invert Iteration to the matrix pencil $(B^{-1}A, I)$ and noting that Step 1 in Algorithm 2 can be implemented as

$$V = (zI - B^{-1}A)^{-1}\tilde{X}^{(i)} = (zB - A)^{-1}B\tilde{X}^{(i)}, \tag{2.65}$$

thereby requiring only a single linear system solution.

2.3.6 Inexact Shift and Invert Iterations

Shift and Invert Iteration can be extended to apply to large eigenvalue problems, where neither factorization-based eigenvalue algorithms nor factorization-based linear system solving algorithms can be used efficiently. This is done by *inexactly* solving the linear systems of equations necessary to implement Step 1 of Algorithm 2 using algorithms for approximating the solutions of linear systems of equations like those

that are described in Section 2.2. The Inexact Shift and Invert Iteration algorithm is described in Algorithm 3.

Algorithm 3 Inexact Shift and Invert Iteration

Inputs:

- $n \times n$ matrix A
- $n \times m$ matrix $\tilde{X}^{(0)}$ whose column vectors are (possibly random) initial guesses for the eigenvectors of the pencil (A, I) whose eigenvalues are closest to some complex scalar z.
- Stopping tolerance ϵ
- Complex scalar shift z located near the eigenvalues that are to be calculated
- Relative tolerance α such that $0 \leq \alpha < 1$, for determining the accuracy of linear system solutions.

For each iteration i:

1. Set $V = \tilde{X}^{(0)}$
2. Orthonormalize V
3. Solve $m \times m$ eigenvalue problem $V^H A V X_v = X_v \tilde{\Lambda}$ for $m \times m$ matrix of eigenvectors X_v and diagonal matrix of eigenvalues $\tilde{\Lambda}$.
4. Set $\tilde{X}^{(i)} = V X_v$
5. Calculate block of residual vectors $R = \tilde{X}^{(i)} \tilde{\Lambda} - A\tilde{X}^{(i)}$. If the norms of all of the column vectors of R are less than ϵ, **stop**. Otherwise **goto** Step 6.
6. Calculate subspace V by solving m linear systems of equations

$$(zI - A)V = \tilde{X}^{(i)} \tag{2.66}$$

   such that the following tolerance on the linear system residuals is met:

$$||\tilde{X}^{(i)} e_j - (zI - A)V e_j|| \leq \alpha \max_{1 \leq t \leq m} ||Re_t|| \quad \forall\ 1 \leq j \leq m \tag{2.67}$$

   where e_j is the j^{th} canonical unit vector.
7. **Goto** Step 2

Outputs: diagonal matrix $\tilde{\Lambda}$ of approximations for the m eigenvalues closest to z, and approximations for the corresponding eigenvectors $\tilde{X}$.

One of the (perhaps surprising) qualities of Inexact Shift and Invert Iteration is that the linear systems of equations do not necessarily need to be solved very accurately [6, 31, 46, 64] in order to ensure fast and robust convergence to the eigenvalues that are near the shift z. Algorithm 3 differs from Algorithm 2 primarily in Step 6 where, in Algorithm 3, the m linear systems of equations are solved such that their

residuals are smaller than the eigenvector residual that is calculated in Step 5. The linear systems of equations need to be solved so that their solution is more accurate than the current solution for the eigenvalue problem, but not necessarily much more accurately than that. Because an estimate for the eigenvector residual is needed for determining the accuracy of the linear system solutions, the Rayleigh-Ritz step is performed first in Algorithm 3, rather than last as in Algorithm 2.

Despite the linear systems of equations being solved inaccurately, Inexact Shift and Invert Iteration still converges linearly, with the rate of convergence being determined by both the eigenvalue spectrum of the eigenvalue problem and by the accuracy of the linear system solutions [64]. Exactly how accurately the linear systems of equations in Step 6 of Algorithm 3 are solved is determined by a user-provided parameter α. Inexact Shift and Invert Iteration will not necessarily converge for all values of α; the largest value of α that will allow for robust convergence depends on the properties of eigenvalue problem, namely the eigenvalue spectrum itself and (for nonsymmetric problems) the eigenvectors [64].

Equation (2.67) in Algorithm 3 indicates that the linear systems of equations in Equation (2.66) must be solved progressively more accurately as the solutions to the eigenvalue problem converge. For most applications, solving a linear system of equations more accurately by using an iterative algorithm means that more iterations of that algorithm must be used in order to calculate the solution. This would seem to imply that the number of iterations (of whatever linear system solving algorithm is being used) that is required for solving Equation (2.66) will grow substantially as Inexact Shift and Invert Iteration converges, which does not bode well for the computational efficiency of Algorithm 3.

In actual practice, however, the number of linear system iterations for solving Equation (2.67) does not increase as the solution to the eigenvalue problem converges, and can actually be quite low (depending on the problem). The precise reason that

this happens depends on the technical details of the linear system solving algorithm; references [64] and [19] discuss this in some detail for the case of GMRES, and a related (but simplified) explanation is provided in Appendix C. In general, though, the reason is somewhat intuitive for algorithms that use Krylov subspaces. For a linear system of equations

$$Ax = b, \tag{2.68}$$

if b is an eigenvector of A such that $Ab = \lambda b$, then the solution to Equation (2.68) is $x = \frac{1}{\lambda}b$. Expressed using matrix polynomials, the solution is $x = p(A)b$ such that $p(A) = \frac{1}{\lambda}I$, i.e. $p(A)$ is a zero-degree matrix polynomial. It thus takes a single iteration of any Krylov linear system algorithm in order to converge to the solution for Equation (2.68). If the right hand side b of Equation (2.68) is an *approximate* eigenvector rather than an exact one, then the degree of the polynomial $p(A)$ must be greater than zero, but it still is likely not very large, depending on how close b is to an exact eigenvector.

As the approximate eigenvectors in Inexact Shift and Invert Iteration become more accurate, Equation (2.66) in Algorithm 3 needs to be solved more precisely, but at the same time it also becomes easier to solve because the right hand sides become better approximations of eigenvectors. The number of Krylov-type that is required to solve it to a given tolerance is thus roughly constant at each inexact shift and invert iteration.

Notably, this does not happen for generalized eigenvalue problems; for generalized eigenvalue problems the number of required linear system iterations increases as the eigenvalue problem solutions converge. This is because Equation (2.66) from Algorithm 3 becomes

$$(zB - A)V = B\tilde{X}^{(i)} \tag{2.69}$$

in the case of generalized eigenvalue problems. In this case, as the approximate eigenvectors $\tilde{X}$ converge to exact eigenvectors X of the pencil (A, B), the right hand

sides of Equation (2.69) converge to BX, which are *not* eigenvectors of the pencil $(zB - A, I)$. The convergence of the generalized eigenvalue problem thus does not help the convergence of the linear system of equations, making the application of Algorithm 3 to generalized eigenvalue problems very inefficient.

This problem can be solved by using by changing Step 6 of Algorithm 3 to be

$$V = \left(\tilde{X}^{(i)} + \delta\tilde{X}\right)(zI - \tilde{\Lambda})^{-1} \tag{2.70}$$

such that $\delta\tilde{X}$ is the solution to the linear system of equations

$$(zB - A)\delta\tilde{X} = -(B\tilde{X}^{(i)}\tilde{\Lambda} - A\tilde{X}^{(i)}). \tag{2.71}$$

This procedure originally comes from reference [31], and the details of its derivation in the context of the FEAST algorithm are provided in Appendix B. It is equivalent to using the special form of linear system restarts that was discussed in Section 2.2.5 (on page 27).

When Equation (2.71) is solved exactly then this procedure is equivalent to solving Equation (2.69) exactly. Using Equation (2.70) has the advantages over Equation (2.69), however, that when Krylov algorithms are used for solving linear systems inexactly. In that case we can recover the property that a roughly constant number of linear system iterations is required at each Inexact Shift and Invert Iteration in order to converge to the desired eigenvectors. Intuitively, this is because the right hand sides of Equation (2.71), which are now the eigenvector residuals for the generalized eigenvalue problem, converge to zero as the approximate eigenvectors and eigenvalues converge to exact eigenvectors and eigenvalues, and the zero vector is an eigenvector of any matrix, including $(zB - A)$ from Equation (2.71).

Inexact Shift and Invert Iteration for generalized eigenvalue problems is described in Algorithm 4.

Algorithm 4 Generalized Inexact Shift and Invert Iteration

Inputs:

- $n \times n$ matrix pencil (A, B)
- $n \times m$ matrix $\tilde{X}^{(0)}$ whose column vectors are (possibly random) initial guesses for the eigenvectors of the pencil (A, B) whose eigenvalues are closest to some complex scalar z.
- Stopping tolerance ϵ
- Complex scalar shift z located near the eigenvalues that are to be calculated
- Relative tolerance α such that $0 \leq \alpha < 1$, for determining the accuracy of linear system solutions.

For each iteration i:

1. Set $V = \tilde{X}^{(0)}$
2. Orthonormalize V
3. Solve $m \times m$ eigenvalue problem $V^H A V X_v = V^H B V X_v \tilde{\Lambda}$ for $m \times m$ matrix of eigenvectors X_v and diagonal matrix of eigenvalues $\tilde{\Lambda}$.
4. Set $\tilde{X}^{(i)} = V X_v$.
5. Calculate block of residual vectors $R = \tilde{X}^{(i)}\tilde{\Lambda} - A\tilde{X}^{(i)}$. If the norms of all of the column vectors of R are less than ϵ, **stop**. Otherwise **goto** Step 6.
6. Solve m linear systems of equations

$$(zB - A)\Delta\tilde{X} = -(B\tilde{X}^{(i)}\tilde{\Lambda} - A\tilde{X}^{(i)}) \tag{2.72}$$

   for $\Delta\tilde{X}$ such that the following tolerance on the linear system residuals is met:

$$||(B\tilde{X}^{(i)}\tilde{\Lambda} - A\tilde{X}^{(i)})e_j - (zB - A)\Delta\tilde{X}e_j|| \leq \alpha \ \ \forall\ 1 \leq j \leq m, \tag{2.73}$$

   where e_j is the j^{th} canonical unit vector. Use the solution $\Delta\tilde{X}$ to form the subspace

$$V = \left(\tilde{X}^{(i)} + \Delta\tilde{X}\right)(zI - \tilde{\Lambda})^{-1} \tag{2.74}$$

7. **Goto** Step 2.

Outputs: diagonal matrix $\tilde{\Lambda}$ of approximations for the m eigenvalues closest to z, and approximations for the corresponding eigenvectors $\tilde{X}$.

2.3.7 Rayleigh Quotient Iteration and Jacobi-Davidson

Rayleigh Quotient Iteration is Shift and Invert Iteration in which the shift is allowed to change at each iteration. It is a naturally single-vector method in which, rather than updating the approximate eigenvectors $\tilde{x}$ as Shift and Invert Iteration does, i.e. solving

$$(zI - A)\tilde{x}^{(i+1)} = \tilde{x}^{(i)}, \tag{2.75}$$

a similar linear system solution is used in which the user-chosen shift z is replaced by the approximate eigenvalue $\tilde{\lambda}$, i.e.

$$(\tilde{\lambda}I - A)\tilde{x}^{(i+1)} = \tilde{x}^{(i)}. \tag{2.76}$$

For exact solutions this improves the rate of convergence compared with Shift and Invert Iteration. Rather than converging linearly, as Shift and Invert Iteration does, Rayleigh Quotient Iteration converges at least quadratically [91].

Also like Shift and Invert Iteration, Rayleigh Quotient Iteration can be used effectively even when Equation (2.76) can only be solved approximately [18, 58, 84]. Rayleigh Quotient Iteration will converge readily by solving Equation (2.76) using a constant number of Krylov subspace iterations at each Rayleigh Quotient Iteration, even though this results in solving Equation (2.76) to the same level of accuracy at every iteration[8].

---

[8]This is where inexact Rayleigh Quotient Iteration and Inexact Shift and Invert Iteration would seem to differ the most. Using a constant number of Krylov linear system iterations at each eigenvalue solving iteration, the accuracy of the linear system solutions improves as the eigenvector solution converges for Shift and Invert Iteration, whereas it stays the same for Rayleigh Quotient Iteration. The key difference is that the conditioning of the linear system of equations in Equation (2.76) becomes worse with each Rayleigh Quotient Iteration; the shift $\tilde{\lambda}^{(i)}$ moves closer to an eigenvalue of A at each Rayleigh Quotient Iteration, making the linear system of equations (2.76) correspondingly more difficult to solve. As noted in [84], this actually does not matter; the approximate solutions for Equation (2.76) move the approximate eigenvector closer to a solution, even though the linear systems of equations themselves never come closer to converging.

Generalized eigenvalue problems $Ax = \lambda Bx$ again pose a special challenge when solving linear systems of equations approximately. For generalized eigenvalue problems Equation (2.76) becomes

$$(\tilde{\lambda}B - A)\tilde{x}^{(i+1)} = B\tilde{x}^{(i)}. \tag{2.77}$$

The right hand side $B\tilde{x}^{(i)}$ is no longer an approximate eigenvector, and so we can expect that using a constant number of Krylov iterations for solving the linear system will result in worse convergence than it would for the standard eigenvalue problem.

In the case of Rayleigh Quotient Iterations, however, it is no longer possible to replace the solution of Equation (2.77) with update equations of the form

$$(zB - A)\delta x = -(\tilde{\lambda}B - A)\tilde{x}^{(i)} \tag{2.78}$$

$$\tilde{x}^{(i+1)} = (\tilde{x}^{(i)} + \delta x)(z - \tilde{\lambda})^{-1}, \tag{2.79}$$

as we did for Inexact Shift and Invert Iterations when deriving Generalized Shift and Invert Iterations. This is because $z = \tilde{\lambda}$ for Rayleigh Quotient Iteration, and so the update equation (2.79) is no longer well-defined.

It is still possible to find an update equation, though, by using a slightly modified approach [109]. Instead of Equation (2.79), we can use

$$\tilde{x}^{(i+1)} = (\tilde{x}^{(i)} + \delta x)\eta^{-1} \tag{2.80}$$

with the scalar η being defined as

$$\eta = \tilde{x}^{(i)H}(\tilde{\lambda}B - A)^{-1}\delta x. \tag{2.81}$$

We can derive a replacement for Equation (2.78) by using the fact that, for exact Rayleigh Quotient Iteration, $\tilde{x}^{(i+1)} = (\tilde{\lambda}B - A)^{-1}B\tilde{x}^{(i)}$ (from Equation (2.77)), and

also the fact that $\tilde{x}^{(i)} = (\tilde{\lambda}B - A)^{-1}(\tilde{\lambda}B - A)\tilde{x}^{(i)}$. Then, starting from Equation (2.80), we have

$$\delta x - \tilde{x}^{(i+1)} = -\tilde{x}^{(i)} \tag{2.82}$$

$$\delta x - \eta(\tilde{\lambda}B - A)^{-1}B\tilde{x}^{(i)} = -(\tilde{\lambda}B - A)^{-1}(\tilde{\lambda}B - A)\tilde{x}^{(i)} \tag{2.83}$$

$$(\tilde{\lambda}B - A)\delta x - \eta B\tilde{x}^{(i)} = -(\tilde{\lambda}B - A)\tilde{x}^{(i)} \tag{2.84}$$

$$(\tilde{\lambda}B - A)\delta x - B\tilde{x}^{(i)}\tilde{x}^{(i)H}(\tilde{\lambda}B - A)^{-1}\delta x = -(\tilde{\lambda}B - A)\tilde{x}^{(i)} \tag{2.85}$$

$$(I - B\tilde{x}^{(i)}\tilde{x}^{(i)H})(\tilde{\lambda}B - A)\delta x = -(\tilde{\lambda}B - A)\tilde{x}^{(i)} \tag{2.86}$$

Thus, when using iterative linear system algorithms for solving generalized eigenvalue problems, we can replace Inexact Rayleigh Quotient Iterations with the update equations

$$(I - B\tilde{x}^{(i)}\tilde{x}^{(i)H})(\tilde{\lambda}B - A)\delta x = -(\tilde{\lambda}B - A)\tilde{x}^{(i)}, \tag{2.87}$$

$$\tilde{x}^{(i+1)} = (\tilde{x}^{(i)} + \delta x)\eta^{-1}. \tag{2.88}$$

The update equations in Equations (2.87) and (2.88) are the same ones that are used in Jacobi-Davidson [85, 87]. Using these update equations without any kind of augmented subspace procedure (as traditional Jacobi-Davidson uses) is sometimes referred to as "Simplified Jacobi-Davidson" or "Newton-Grassman" [18, 90] (because of their relationship to Newton-type methods). The parameter η in Equation (2.88) is set 1 in actual practice, because it is the direction of $\tilde{x}^{(i+1)}$ that matters when solving an eigenvalue problem, and not the magnitude. Although we motivated the derivation of the Jacobi-Davidson update equations by trying to solve generalized eigenvalue problems, they are most often used for solving standard eigenvalue problems, where they can have some advantages over Rayleigh Quotient Iteration in terms of robustness [105].

This way of deriving the Jacobi-Davidson update equations may seem a bit contrived, and that is because it is. It is very unlikely that one would think to set $\eta = \tilde{x}^{(i)H}(\tilde{\lambda}B - A)^{-1}\delta x$ when starting from Equation (2.80). My purpose in doing it this way is to emphasize that the relationship between Rayleigh Quotient Iteration and Jacobi-Davidson is essentially the same as the relationship between Shift and Invert Iteration and Generalized Shift and Invert Iteration; the only difference is that a different version of iterative refinement is used in solving the linear systems of equations. The projector $(I - B\tilde{x}x^H)$ that is so characteristic of Jacobi-Davidson is necessarily primarily because the shift in Rayleigh Quotient Iteration is the same as the Ritz value for $\tilde{x}^{(i)}$.

2.3.8 Augmented Subspace Methods

With the exception of Arnoldi and Lanczos iterations, the algorithms discussed in this chapter - and in forthcoming chapters - operate using subspaces of static dimension. All of the methods described in this dissertation (both in the preceding sections and in later chapters) can, however, also be combined with some method for subspace expansion in order to enhance their convergence rates. For example, Shift and Invert Iteration is the same as Subspace Iteration applied to the matrix $(zI-A)^{-1}$. We could, instead, use Lanczos or Arnoldi iterations with the matrix $(zI - A)^{-1}$, in which case the dimension of the subspace would grow with each iteration, and the estimated eigenvalues and eigenvectors would converge more quickly as a result.

I will lump all such algorithms together as "augmented subspace" methods, and I will consider them to be largely beyond the scope of this dissertation, which focuses on methods that use statically-sized subspaces. Several notable algorithms fall into the category of augmented subspace methods. The standard Jacobi-Davidson algorithm [85, 87] is a modification of the Simplified Jacobi-Davidson algorithm (described in the previous section), wherein the eigenvector approximation subspace is augmented

with the updated approximation from Equation (2.88) at each iteration, rather than being replaced by it. Rational Krylov [67] uses Shift and Invert Iteration to generate an expanding subspace, changing the value of the shift at each iteration. The methods of Sakurai and Seguira [39, 76, 77] apply contour integral-based filtration (see Section 3.1 on page 60) to Krylov subspaces, and were the first methods to effectively use contour integral-based spectral slicing. The FEAST algorithm, which is the primary subject of this dissertation, can also be used with augmented subspaces [24, 25].

2.4 Challenges with large eigenvalue problems

2.4.1 Solving for too many eigenpairs produces bad scaling

The algorithms for solving eigenvalue problems that are discussed in Section 2.3 all attempt to solve an eigenvalue problem by first identifying the subspace that contains the eigenvectors of interest, and then projecting the original eigenvalue problem in to that subspace, thereby reducing its dimension. From there, the resulting reduced eigenvalue problem can be solved by using an algorithm that diagonalizes dense matrices, such as QR iterations.

The algorithmic complexity of actually solving for the eigenpairs of interest is thus reduced from being the cube of the original dimension [30], $O(n^3)$, which is much too computationally challenging, to being the cube of the dimension of the subspace of interest, $O(m^3)$. A computational complexity of $O(m^3)$ is tractable for many common applications, particularly those that require finding a small number of the largest-magnitude eigenvalues; in this case the subspace dimension m is small enough that the reduced eigenvalue problem can be solved rapidly. The most computationally intensive task instead becomes the calculation of a basis for the subspace of interest, which is where eigenvalue algorithms spend most of their effort.

The situation changes when we want to find a large number of eigenpairs. Applications like this occur often in electronic structure theory, which requires the calculation

of a number of eigenvalues that is equal to the number of electrons that is being simulated. Simulating a single carbon atom thus requires the calculation of 6 eigenvalues; simulating a large collection of organic molecules that are in a solution of water molecules, on the other hand, requires the calculation of many thousands of eigenvalues. The dimension of the eigenvector subspace must always be at least as large as the number of eigenvalues that we seek to calculate, and so the "reduced" eigenvalue problems for such calculations can quickly become a bottleneck for effectively scaling to larger problem sizes.

A similar situation occurs in applications that require the calculation of eigenvalues that are in the interior of the spectrum. Because traditional Subspace Iteration or Krylov methods can only calculate exterior eigenvalues, then the only way to identify interior eigenvalues is to calculate a sufficiently large number of eigenpairs that both the (undesired) exterior eigenvalues and the (desired) interior ones are captured in the same subspace. The result is that a very large subspace is required for the calculation of even a small number of interior eigenpairs, leading to the aforementioned problem of having a reduced eigenvalue problem that is, itself, still too large.

Shift and Invert Iteration can help with this problem; it allows one to calculate the eigenvalues that are located near some shift z in the complex plane. It introduces challenges of its own, however. In order to achieve a good rate of convergence when using Shift and Invert Iteration, it is best to locate the shift z as close as possible to the eigenvalue of interest; locating the shift close to an eigenvalue increases the difficulty of performing the inverse multiplications $(zI - A)^{-1}x$ by making the conditioning of $(zI - A)$ worse, though. Moreover, it is still challenging to calculate large numbers of eigenpairs, as only the ones that are very close to the shift will converge quickly.

What is needed for calculating large numbers of eigenpairs, or for calculating eigenpairs with eigenvalues in the interior of the spectrum, is a method for selectively calculating only the eigenpairs whose eigenvalues are in a specific region in the com-

plex plane. Eigenvalue algorithms that can do this are said to be performing "spectral slicing".

2.4.2 A solution: spectral slicing

"Spectral slicing" is a process whereby the complex plane is divided into multiple, user-determined disjoint regions, and the eigenpairs belonging to each region are calculated independently of the eigenpairs in the other regions. The ability to do this has immediate benefits for calculating large numbers of eigenpairs: rather than calculating (for example) 10,000 eigenpairs by using a subspace of dimension $m = 10,000$, one can instead calculate 10,000 eigenpairs by dividing the complex plane into 100 disjoint regions with 100 eigenvalues inside of each one, and then using a subspace dimension of $m = 100$ for calculating the eigenpairs in each region. Because of the $O(m^3)$ scaling of the Rayleigh-Ritz process, this can potentially result in a much faster calculation, even if the eigenvalue problems the disjoint regions in the complex plane are solved one after another in a sequence.

That implies yet another immediate benefit of spectral slicing: by treating these disjoint regions in the complex plane independently of each other, large numbers of eigenpairs can be calculated simultaneously in parallel, rather than sequentially. Thus $10,000$ eigenpairs can potentially be calculated in the same amount of time that is required to calculate 100 eigenpairs, provided that we have 100 separate computers to run the calculations on.

Spectral slicing is performed by replacing the original matrix A with a new matrix $f(A)$, where $f(z) : \mathbb{C} \to \mathbb{C}$ is analytic[9] for scalar z. One then uses either Subspace Iteration or Krylov methods to calculate the largest-magnitude eigenvalues of the

[9]The analyticity of the function matters because, if $f(z)$ is analytic, then the matrix $f(A)$ has the same eigenvectors as A, and each of its eigenvectors x_i has the corresponding eigenvalue $f(\lambda_i)$, where λ_i is the original eigenvalue of A. We can thus change the eigenvalues of $f(A)$ to be almost anything we want by judiciously choosing $f(z)$.

matrix $f(A)$ instead. A different function $f_i(\lambda)$ is chosen for each disjoint region in the complex plane $\mathcal{R}_i$ such that each $f_i(z)$ is large for values of z that are inside the region $\mathcal{R}_i$, and is small everywhere else. Because $f_i(z)$ takes its largest values in the region $\mathcal{R}_i$, the eigenvector subspace for the largest magnitude eigenvalues of $f(A)$ is the same as the eigenvector subspace for the eigenvalues of A that are in the region $\mathcal{R}_i$. Thus we can calculate the eigenvectors of A for only the eigenvalues in the region $\mathcal{R}_i$ by instead calculating the eigenvectors for the largest magnitude eigenvalues of $f_i(A)$, a task that can be accomplished by using any of the previously-mentioned eigenvalue algorithms. The generic spectral slicing process is summarized in Algorithm 5.

Algorithm 5 Spectral Slicing

1. Choose $f(\lambda)$ that is large for the eigenvalues of interest, and small everywhere else in $\mathbb{C}$.
2. Calculate a subspace basis V that spans the m dominant eigenvectors X_1 of $f(A)$.
3. Solve the reduced eigenvalue problem $V^H A V = \lambda V^H V x_v$ for the eigenvalues of interest.

One particularly common and straight-forward choice for $f(z)$, and the one that I will be focusing on in this dissertation, is the indicator function,

$$f(z) = \begin{cases} 1, & z \in \mathcal{R} \\ 0, & \text{otherwise.} \end{cases} \tag{2.89}$$

If the eigendecomposition of A is divided into two parts,

$$A = X_{R1}\Lambda_1 X_{L1}^H + X_{R2}\Lambda_2 X_{L2}^H, \tag{2.90}$$

such that the diagonal elements of Λ_1 are the eigenvalues that are inside the region $\mathcal{R}$ (with the corresponding right and left eigenvectors being the column vectors of X_{R1} and X_{L1}), and the diagonal elements of Λ_2 are all of the other eigenvalues of A, then

$$f(A) = X_{R1} f(\Lambda_1) X_{L1}^H + X_{R2} f(\Lambda_2) X_{L2}^H = X_{R1} I X_{L1}^H + X_{R2} 0 X_{L2}^H = X_{R1} X_{L1}^H. \quad (2.91)$$

Choosing $f(z)$ to be the indicator function over the region $\mathcal{R}$ results in $f(A)$ being a spectral projector for the eigenvectors of A whose eigenvalues are inside that region.

If Subspace Iteration is used with this choice of $f(A)$, then convergence to the desired eigenvector subspace occurs in a single iteration: multiplying (almost) any subspace basis by $f(A) = X_{R1} X_{L1}^H$ projects it on to exactly the subspace for the desired eigenvectors, and the Rayleigh-Ritz method produces exactly the desired eigenvalues as a result. This sounds too good to be true and, of course, it is. The only way to evaluate $f(A)$ exactly when $f(z)$ is the indicator function is to find the eigenvalue decomposition of A and then form the outer product $X_{R1} X_{L1}^H$ explicitly, which obviously does not help us if our goal is to use $f(A)$ to help calculate the eigenvalue decomposition in the first place.

Instead, practical spectral slicing algorithms form some sort of computationally tractable approximation for the indicator function of a matrix. The focus of this dissertation concerns one particular choice of approximation, which is to use line integrals in the complex plane around the region of interest.

CHAPTER 3

THE FEAST ALGORITHM: SPECTRAL SLICING WITH CONTOUR INTEGRATION AND SUBSPACE ITERATIONS

Solving eigenvalue problems by using spectral slicing requires choosing a matrix function $f(A)$ with which to perform the slicing, and choosing an underlying eigenvalue algorithm use with the matrix $f(A)$. In this dissertation I consider the algorithms that are produced when we choose $f(A)$ to be a Cauchy integral-based representation of the indicator function, and the underlying eigenvalue algorithm to be Subspace Iteration. These choices of matrix function and eigenvalue algorithm are what broadly define the FEAST algorithm.

The standard FEAST algorithm [63, 93, 94] is the result of replacing the Cauchy integral with a quadrature rule approximation, while still evaluating the integrand exactly. In the following sections I discuss the behavior of, and the theory behind, the standard FEAST algorithm.

3.1 Spectral Slicing with Cauchy Integrals

Section 2.4.2 discusses the process of spectral slicing, wherein, rather than using Subspace Iteration for directly solving an eigenvalue problem with the matrix A, one instead uses Subspace Iteration for solving an eigenvalue problem with the matrix $f(A)$. The function $f(z)$ is chosen such that the dominant eigenvector subspace[1] of $f(A)$ corresponds exactly to the eigenvector subspace of A for the eigenvalues that lie

[1] I.e. the subspace spanned by the eigenvectors with the largest-magnitude eigenvalues.

inside of a particular, user-defined contour $\mathcal{C}$ in the complex plane that encloses the region $\mathcal{R}$ of interest. This way one can solve for eigenvector subspaces corresponding to eigenvalues in different regions of the complex plane independently of each other, making it possible to parallelize the solution of eigenvalue problems and reduce the severity of the $O(m^3)$ scaling of the Rayleigh-Ritz process, where m is the number of eigenvalues that are in the region $\mathcal{R}$.

A common choice for $f(z)$ is the indicator function,

$$f(z) = \begin{cases} 1, & z \in \mathcal{C} \\ 0, & \text{otherwise,} \end{cases} \tag{3.1}$$

which is equal to one in the region that contains the m eigenvalues of interest, and zero everywhere else. Evaluating the indicator function of a matrix, $f(A)$, exactly requires knowing the eigenvalue decomposition of that matrix. This is what we seek to calculate in the first place, and so practical eigenvalue algorithms use approximations for $f(A)$ instead. This dissertation focuses on a particular class of algorithms that approximate $f(A)$ by using contour integration in the complex plane.

The indicator function in Equation 3.1 can be written exactly by using Cauchy's integral formula,

$$f(\lambda) = \frac{1}{2\pi i} \oint_{\mathcal{C}} (z - \lambda)^{-1} dz = \begin{cases} 1 & \lambda \in \mathcal{C} \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

where the integration is performed over the closed contour $\mathcal{C}$. To apply this function to a matrix, the input λ is replaced by the matrix A and the integration variable z is replaced by the diagonal matrix zI,

$$f(A) = \frac{1}{2\pi i} \oint_{\mathcal{C}} (zI - A)^{-1} dz. \tag{3.3}$$

Eigenvalue solving algorithms can thus be developed that use the matrix vector products

$$f(A)X = \frac{1}{2\pi i} \oint_{\mathcal{C}} (zI - A)^{-1} X dz \tag{3.4}$$

in place of AX. Although this dissertation focuses on using Subspace Iteration with this matrix vector product, it is possible to use Cauchy integrals with other kinds of eigenvalue algorithms as well [9, 39, 76, 77].

Because the eigenvalue decomposition of A is assumed to be unknown, the integration in Equation (3.4) must be approximated somehow. There are two operations that can be approximated in order to make the evaluation of (3.4) tractable: the integration itself, and the application of the matrix inverse in the integrand. Depending on which of these operations is approximated, and how the approximation is performed, one arrives at one of several different contour integration-based algorithms for solving eigenvalue problems.

The rest of the sections in this chapter discuss the algorithm that is produced by approximating only the integration operation.

3.2 The Standard FEAST Algorithm

The standard FEAST algorithm is a spectral slicing algorithm that performs approximate subspace iterations with $f(A)$ by approximating the matrix vector product

$$f(A)X = \frac{1}{2\pi i} \oint_{\mathcal{C}} (zI - A)^{-1} X dz \tag{3.5}$$

with a quadrature rule

$$\rho(A)X = \sum_{k=1}^{n_c} \omega_k (z_k I - A)^{-1} X \approx f(A)X. \tag{3.6}$$

The constants ω_k are the quadrature weights (which I take to include the $\frac{1}{2\pi i}$ term), and the constants z_k are the quadrature points. In the case of generalized eigenvalue problems $Ax = \lambda Bx$, we can replace the matrix A in equation (3.6) with $B^{-1}A$ to get the matrix vector product

$$\rho(B^{-1}A)X = \sum_{k=1}^{n_c} \omega_k (z_k I - B^{-1}A)^{-1} X \tag{3.7}$$

$$= \sum_{k=1}^{n_c} \omega_k (z_k B - A)^{-1} BX \equiv \rho(A, B)X. \tag{3.8}$$

I will often use the notation $\rho(A, B)$ in place of $\rho(B^{-1}A)$, to emphasize that no linear systems of equations are solved with the matrix B. The standard FEAST algorithm, then, is itself a variation of Subspace Iteration that uses the matrix $\rho(A, B) = \sum_k \omega_k (z_k B - A)^{-1} B$.

In practice the matrix vector products $\rho(A, B)X$ are calculated by directly solving a collection of n_c linear systems of equations

$$(z_k B - A)Y_k = BX \tag{3.9}$$

for the unknowns Y_k and adding their solutions together in a weighted sum

$$Q = \rho(A, B)X = \sum_{k=1}^{n_c} \omega_k Y_k. \tag{3.10}$$

This is the reason that we use equation (3.8) rather than equation (3.7); calculating $\rho(A, B)X$ with the matrices $(z_k B - A)^{-1} B$ requires the solution of only a single linear system of equations per quadrature shift z_k, whereas using the matrices $(z_k I - B^{-1}A)^{-1}$ also requires the calculation of the matrix $B^{-1}A$, which can be prohibitively expensive. It is worth noting that using equation (3.8) changes the right hand sides of the linear systems from being approximate eigenvectors, X, to being a matrix product

of approximate eigenvectors, BX. This will have important implications in Chapter 4, where we will consider whether and how FEAST should be implemented if we are only able to solve the linear systems in equation (3.9) inexactly.

The key assumption in the FEAST algorithm is that one is able to efficiently calculate exact solutions to the linear systems of equations (3.9). The open source FEAST software package [62], for example, does this by using the PARDISO [78] solver, which solves sparse linear systems of equations by efficiently performing factorizations of the coefficient matrix. In general, if the FEAST coefficient matrix $(z_k B - A)$ can not be factorized efficiently by some means, then the linear systems of equations (3.9) can not be solved exactly in an efficient manner, and so the associated eigenvalue problem can not be solved by using the standard FEAST algorithm.

Algorithm 6 describes a basic version of the standard FEAST algorithm. Most of the computation in the algorithm occurs in **Step 1**, where n_c shifted linear systems are solved. One of the benefits of the FEAST algorithm is that linear systems with different values of z_k are independent of each other, and so all of these linear systems can be solved simultaneously in parallel. With enough available parallel computing power, then, each FEAST subspace iteration can be performed in the amount of time that is required to solve a singe linear system right hand side. As a result, because the rate of convergence improves with increased n_c (see Section 3.3), the FEAST algorithm can be used to solve eigenvalue problems very rapidly, especially when a lot of parallel computing power is available.

Algorithm 6 is a simplified version FEAST; it will work well for many problems, but it omits several implementational details that can improve the speed and robustness of the algorithm. Some of these details include:

- orthogonalizing the subspace Q in a rank-revealing way by using the QR decomposition or SVD before performing the Rayleigh-Ritz procedure in **Step 3**

Algorithm 6 The FEAST algorithm for solving $AX = BX\Lambda$

Inputs:

- Matrices $A, B \in \mathbb{C}^{n \times n}$
- Closed contour $\mathcal{C}$ that encloses the search region for eigenvalues in the complex plane
- Overestimate m_0 for the number of eigenvalues inside $\mathcal{C}$ [a]
- Initial guess $\tilde{X}^{(0)} \in \mathbb{C}^{n \times m_0}$ for the search subspace spanned by the solution to the eigenvalue problem
- Set of n_c quadrature weights and points (ω_k, z_k) for numerically integrating equation (3.5) [b]

For each subspace iteration i:

1. Directly solve n_c shifted linear systems, each with m_0 right hand sides, for $Y_k^{(i)} \in \mathbb{C}^{n \times m_0}$.

$$\frac{1}{\omega_k}(z_k I - A)Y_k^{(i)} = \tilde{X}^{(i)}, \quad 1 \leq k \leq n_c$$

2. Form the filtered subspace Q

$$Q = \hat{\rho}(A)\tilde{X}^{(i)} = \sum_{k=1}^{n_c} Y_k^{(i)}$$

3. Perform Rayleigh-Ritz procedure to find a new estimate for eigenvalues and eigenvectors:

i. Solve the generalized reduced eigenvalue problem for $X_Q \in \mathbb{C}^{m_0 \times m_0}$ and $\tilde{\Lambda} \in \mathbb{C}^{m_0 \times m_0}$

$$A_Q X_Q = B_Q X_Q \tilde{\Lambda}$$

with $A_Q = Q^H A Q$ and $B_Q = Q^H Q$

ii. Get new estimate for subspace $\tilde{X}^{(i+1)}$: $\tilde{X}^{(i+1)} = Q X_Q$

4. Calculate the FEAST eigenvector residual $||R_F|| = max\ ||\tilde{\lambda}_j B\tilde{x}_j^{(i+1)} - A\tilde{x}_j^{(i+1)}||$, $1 \leq j \leq m_0$, $\tilde{\lambda}_j$ inside $\mathcal{C}$. If $||R_F||$ is above a given tolerance, **GOTO 1**.

Outputs: diagonal matrix $\tilde{\Lambda}$ of approximations for the m eigenvalues inside $\mathcal{C}$, and approximations for the corresponding eigenvectors $\tilde{X}$

---

[a] Overestimation is important; if m_0 is smaller than the actual number of eigenvalues m that are enclosed by $\mathcal{C}$ then the algorithm will fail to converge.

[b] Any quadrature rule can be used, e.g. Gaussian quadrature, trapezoidal or Zolotarev rule [34].

can help to prevent the occurrence of "spurious" eigenvalues, wherein FEAST calculates eigenvalues that do not exist for the original matrix A;

- performing rank-revealing orthogonalizations of Q can also allow one to dynamically reduce the dimension of that subspace; the solution of the reduced eigenvalue problem in **Step 3(i)** can produce an error in eigenvalue solving packages (such as LAPACK [1]) when Q is rank-deficient;

- simultaneously calculating both the left and the right eigenvectors for a non-Hermitian eigenvalue problem, and biorthogonalizing them in between iterations, can substantially improve the robustness of the algorithm [93];

- when solving Hermitian eigenvalue problems it is possible to use the reflection symmetry of the eigenvalue spectrum across the real axis of the complex plane in order to reduce by half the number of linear systems that need to be solved in **Step 1**, as is done in the original FEAST paper [63].

These details, as well as other features, are implemented in the FEAST software package [62]. A description of the two-sided Inexact FEAST algorithm (which incorporates ideas from Chapter 4) is also provided in Appendix D (on page 189), which shows how biorthogonalization can be used for solving nonsymmetric problems.

3.3 Convergence

The standard FEAST algorithm is a variation of standard Subspace Iteration, using the matrix products $\rho(A, B)X$ from Equation (3.8) in place of $B^{-1}A$, and so it converges linearly just as standard Subspace Iteration does. The FEAST equivalent of Inequality (2.62) (on page 41) is

$$||\tilde{q}_j - x_j|| \leq \left(\frac{\gamma_{m_0+1}}{\gamma_j}\right) ||q_j - x_j||, \tag{3.11}$$

where $||q_j - x_j||$ is an upper bound on the eigenvector error for eigenvector j at the current FEAST iteration, $||\tilde{q}_j - x_j||$ is an upper bound on the error for the same eigenvector at the next FEAST iteration, and γ_k denotes the k^{th} eigenvalue of the operator $\rho(B^{-1}A)$ from equation (3.8). The rate of convergence for FEAST, which is

$$\left(\frac{\gamma_{m_0+1}}{\gamma_j}\right), \tag{3.12}$$

takes the same form as it does for standard subspace iterations, except now λ_k, the eigenvalues of $B^{-1}A$, are replaced by γ_k, the eigenvalues of $\rho(B^{-1}A)$. Note that the relationship between γ_k and λ_k is just $\gamma_k = \rho(\lambda_k)$, where the λ_k are ordered such that $\gamma_k > \gamma_{k+1}$. FEAST thus acts as a filter for the original eigenvalues λ_k of the pencil (A, B), mapping the values of λ_k that are inside of $\mathcal{C}$ to be the dominant eigenvalues of the matrix $\rho(A, B)$. The function $\rho(z)$ is itself a rational filter function that selects for the values of z that are inside of the contour $\mathcal{C}$.

The rate of convergence of the FEAST algorithm is at its largest when the ratio $\gamma_{m_0+1}/\gamma_j = \rho(\lambda_{m_0+1})/\rho(\lambda_j)$ is very large, meaning that the eigenvalues we don't want (including the eigenvalue λ_{m_0+1}) are mapped by $\rho(z)$ to approximately zero, and the eigenvalues that we do want (i.e. the λ_j) are mapped by $\rho(z)$ to approximately one. The two FEAST parameters with the strongest impact on the rate of convergence are the dimension of the FEAST subspace, m_0, and the number of quadrature points used in approximating the contour integration, n_c.

Increasing m_0 changes λ_{m_0+1} to be an eigenvalue of $B^{-1}A$ that is farther away from the eigenvalues that we are interested in calculating. Because $\rho(z)$ is an approximation of the indicator function for the region of the complex plane enclosed by the contour $\mathcal{C}$, it becomes progressively smaller as z is moved farther away from the region enclosed by $\mathcal{C}$; as a result, if λ_j is inside $\mathcal{C}$, then the ratio $\rho(\lambda_{m_0+1})/\rho(\lambda_j)$ decreases as m_0 is increased. The following section, Section 3.4, offers a simple illustration of this.

Increasing the number of quadrature points, n_c, has the effect of improving the accuracy of the approximation of the integral in equation (3.4). This brings $\rho(z)$ closer to being $\rho(z) = 1$ for values of z inside of $\mathcal{C}$, and closer to being $\rho(z) = 0$ for values of z outside of $\mathcal{C}$. Improving the quadrature approximation of equation (3.4) can have substantial benefits even with only modest increases in the value of n_c; the trapezoidal quadrature rule, for example, converges geometrically to the value of the exact integral for a circular contour [100]. Other choices for the quadrature rule can produce even better performance, depending on the spectrum of the matrix that is being diagonalized. Gauss quadrature, for example, can improve the performance of the FEAST algorithm when the reflection symmetry across the real number line is being used in calculating the eigenvalues of Hermitian matrices, and a Zolotarev quadrature rule can allow for reliable convergence even when the spectrum of a Hermitian eigenvalue problem might otherwise cause convergence to be extremely difficult [34].

Importantly for modern computing applications, both of these methods of improving the convergence rate of the standard FEAST algorithm are embarrassingly parallel. All of the m_0 right hand sides of each of the linear systems of equations (3.9) can be solved independently of each other, and all of the n_c linear systems of equations can be solved independently of each other as well. The convergence rate of the FEAST algorithm can thus be systematically improved just by using more parallel processing for solving a larger number of linear systems of equations.

3.4 A Simple Example

The convergence behavior of FEAST is most easily understood by examining the action of the filter $\rho(z)$ on complex scalars z, and especially on the eigenvalues of a particular matrix pencil that one wants to diagonalize. I illustrate this with a simple example by using FEAST to calculate some of the eigenvalues and eigenvectors of a 12×12 Hermitian matrix A.

Figure 3.1 illustrates the FEAST search contour for A, and Figure 3.2 shows two examples of filter functions $\rho(z)$ that are produced by discretizing and evaluating the FEAST contour integral. The search contour selects the middle 4 eigenvalues of A. The resulting filter functions are approximately equal to 1 inside of the contour, particularly when they are evaluated at the desired 4 eigenvalues of A, and their magnitudes quickly reduce outside of the search contour. Using larger numbers of quadrature points n_c causes the FEAST filter function to reduce more quickly outside of the search contour, thereby increasing the rate of convergence to a solution.

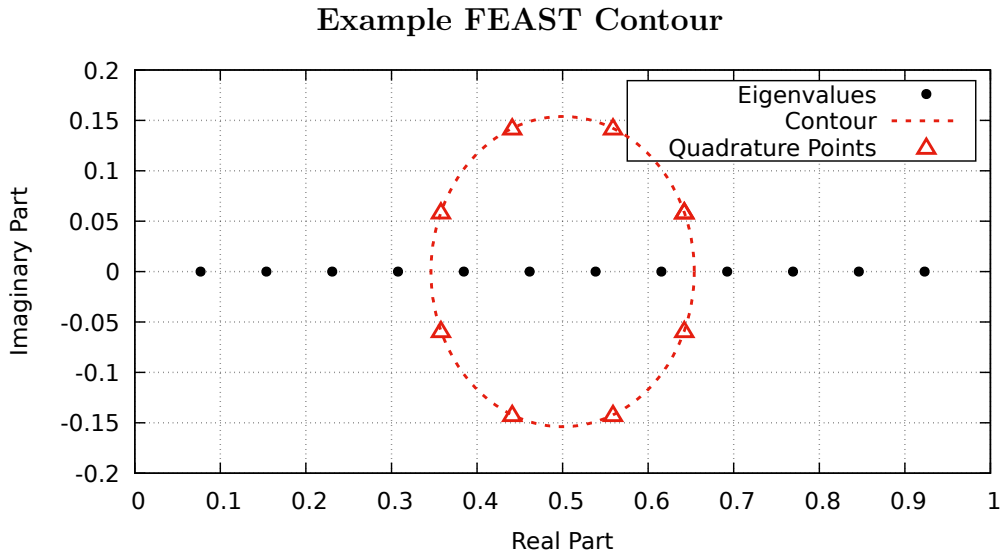


Figure 3.1. *Illustration of the FEAST search contour for a* 12×12 *Hermitian matrix. The eigenvalues are distributed uniformly on the real number line, and a closed contour is used to select 4 of them in the center of the spectrum. Also depicted are the locations of the quadrature points when* $n_c = 8$ *and the trapezoidal rule is used to discretize the contour integration.*

The filter functions that are depicted in Figure 3.2 can be used to provide a quantitative estimate of the FEAST rate of convergence from Equation (3.12), if we plot them in a slightly different way. Figure 3.3 shows the filtered eigenvalues from Figure 3.2 plotted in sorted order by magnitude. The value of the $(m_0 + 1)^{\text{th}}$ largest-magnitude filtered eigenvalue, divided by $\gamma_4 = \rho(\lambda_4) \approx 1$, gives the rate of convergence for that value of m_0. Two such example values are indicated in the

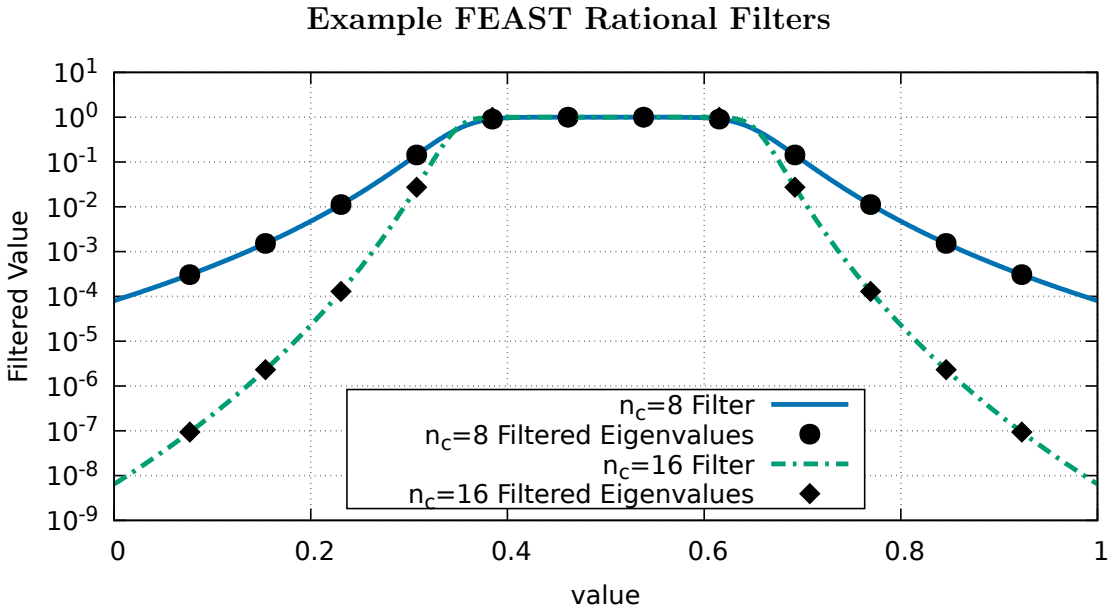


Figure 3.2. *Plot of the value of the filter $\rho(z)$ on the real number line for the contour depicted in Figure 3.1. Plot shows the value of the filter function for $n_c = 8$ and $n_c = 16$; larger values of n_c make the value of $\rho(z)$ decrease more quickly outside of search contour.*

plot in Figure 3.3. Notably, Figure 3.3 indicates that when $m_0 < 4$ the error in the eigenvalue estimation is not reduced by performing FEAST iterations.

Figure 3.4 shows plots of the eigenvector residual versus the FEAST iteration number when using FEAST calculate the eigenvalues and eigenvectors of A. The FEAST rate of convergence is the slope of the lines in Figure 3.4; these slopes can be predicted by taking the logarithms of the filtered values depicted in Figure 3.3. For the standard FEAST algorithm (and indeed for all subspace iteration algorithms) these values typically match very closely. Note, for example, that the logarithm of the filter value for "$m_0 + 1 = 4$" in Figure 3.3 is 0; the slope of the line in Figure 3.4 for $m_0 = 3$ is also approximately 0. This illustrates the importance of over-estimating the number of eigenvalues inside of the FEAST contour: if that number is under-estimated instead, then FEAST will not converge to a solution.

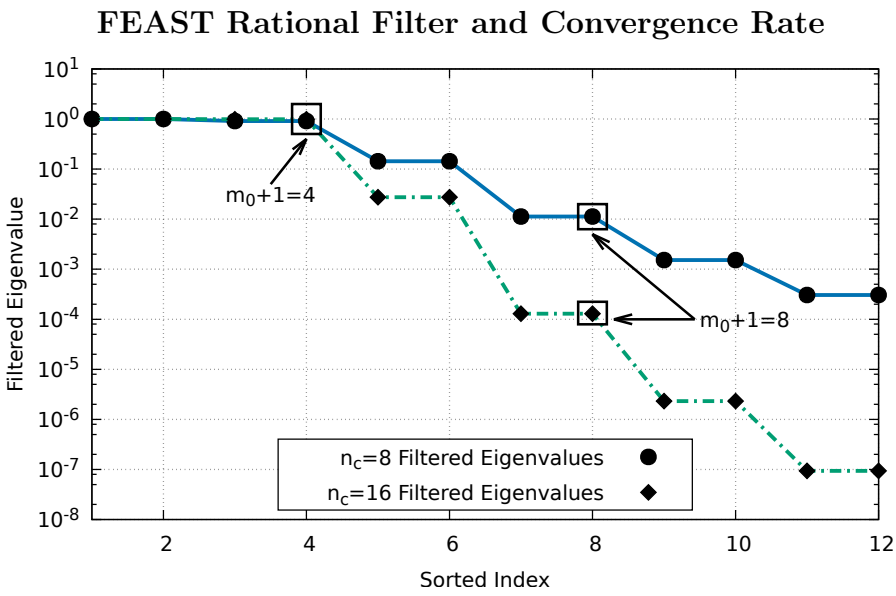


Figure 3.3. *The filtered eigenvalues from Figure 3.2, this time plotted in sorted order, from largest magnitude to smallest magnitude. Boxes and text labels indicate the* $(m_0+1)^{th}$ *largest-magnitude filtered eigenvalues for* $m_0 = 3$ *and* $m_0 = 7$*. The value of* $\rho(\lambda_{m_0+1})/\rho(\lambda_4)$ *at these eigenvalues determines the convergence rate (i.e. Equation (3.12)) of FEAST for subspace dimensions* $m_0 = 3$ *and* $m_0 = 7$*, respectively; lower values indicate that the error in the eigenvector estimation is reduced more quickly.*

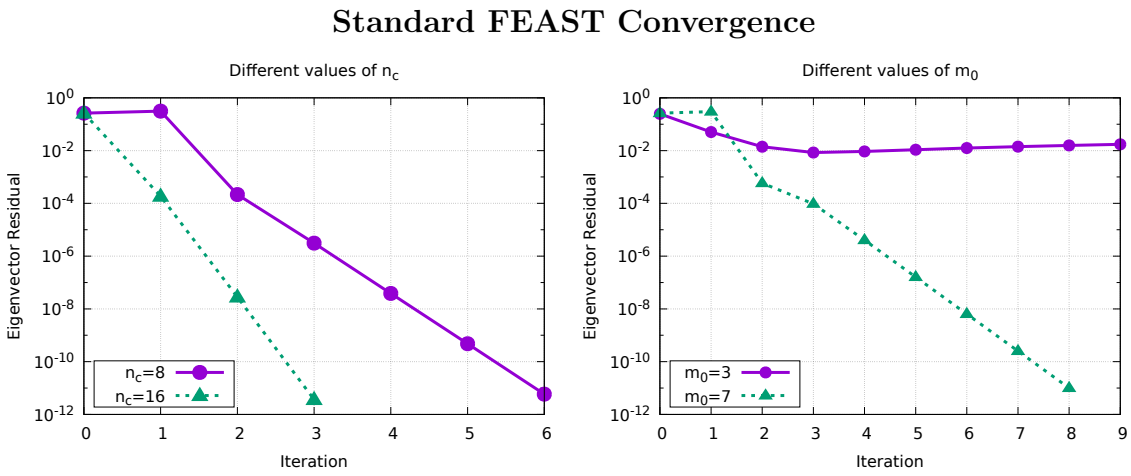


Figure 3.4. *Plots showing the eigenvector residual versus FEAST iteration number for several values of the parameters* n_c *(i.e. number of quadrature points) and* m_0 *(i.e. FEAST subspace dimension). The plotted eigenvector residual is the largest residual for all of the estimated eigenvectors whose eigenvalues are inside of the search contour. The left plot uses two different values of* n_c *with* $m_0 = 7$*. The right plot uses two different values of* m_0 *with* $n_c = 8$*.*

CHAPTER 4

THE IFEAST ALGORITHM: FEAST WITH APPROXIMATE SOLVES

The standard FEAST algorithm, as described in Chapter 3, is a spectral slicing algorithm that uses a Cauchy integral-based approximation for the indicator function in conjunction with Subspace Iteration in order to calculate the eigenvectors whose eigenvalues lie inside of a particular, user-defined contour $\mathcal{C}$ in the complex plane. It has the benefit of being a naturally parallelizable algorithm, in the sense that the speed with which it performs calculations can be systematically improved by increasing the amount of computation that is done in parallel.

The FEAST algorithm works by performing Subspace Iteration with the matrix

$$\rho(A,B)X = \sum_{k=1}^{n_c} \omega_k (z_k B - A)^{-1} BX, \tag{4.1}$$

which is an approximation for the Cauchy integral

$$f(A,B)X = \frac{1}{2\pi i} \oint_{\mathcal{C}} (zB - A)^{-1} BX dz \tag{4.2}$$

in which the integration operation is approximated by a quadrature rule. The matrix vector products $(z_k B - A)^{-1} BX$ are calculated directly by solving linear systems of equations

$$(z_k B - A) Y_k = BX \tag{4.3}$$

for Y_k by using factorization-based methods.

Because it relies on factorization-based methods for solving linear systems of equations, the standard FEAST algorithm can only be applied to eigenvalue problems $Ax = \lambda Bx$ whose matrices A and B can be factorized efficiently. As discussed in Section 2.4, however, there are many eigenvalue problems where efficient matrix factorization is not possible, either because the dimension of the matrices is too large or because the matrices are not explicitly stored at all. In cases like these, the linear systems of equations (4.3) can only be solved approximately, usually by using algorithms that rely on matrix vector multiplication (prominent examples of which are discussed in Section 2.2).

The Inexact FEAST algorithm (which I will often refer to as IFEAST) is a modification of the FEAST algorithm such that its linear systems of equations are solved inexactly by using some sort of iterative algorithm. This allows the FEAST framework to be applied to solving eigenvalue problems that are too large to be solved with the standard FEAST algorithm.

The following sections describe the IFEAST algorithm in detail and explain the theory behind its convergence behavior. Where the standard FEAST algorithm is based on rational filter functions, IFEAST is based on polynomial filter functions. We will find that the IFEAST algorithm retains some of the desirable qualities of the standard FEAST algorithm, but its ability to be parallelized is limited by the accuracy with which its linear systems of equations are solved, and that this is related to the limitations of using polynomial filters for solving eigenvalue problems.

4.1 The Inexact FEAST Algorithm

Modifying the FEAST algorithm such that it converges successfully when iteratively solving the linear systems in Equation (4.3) can be trivially straight-forward if done naively; if the linear systems are solved to high accuracy, e.g. such that their relative residuals are on the order of 10^{-15}, then of course one can reasonably expect

that the solutions to the eigenvalue problem will converge as they do normally when using the standard FEAST algorithm. The problem with this approach is that it is usually very prohibitively expensive. Solving most linear systems of equations to such a high precision by using an iterative algorithm will often require a number of iterations that is on the order of the dimension of the eigenvalue problem, and the solution of an eigenvalue problem with FEAST typically requires the solution of several such linear systems of equations.

Instead, the IFEAST algorithm attempts to solve linear systems of equations no more accurately than is strictly necessary to ensure convergence. This dissertation will discuss two methods for doing this. One of these methods is a relatively simple modification of the standard FEAST algorithm that allows for the efficient solution of standard eigenvalue problems by using iterative linear system solving methods: at each FEAST iteration, the linear systems of equations are solved such that their their residuals are some user-determined fraction of the current eigenvector residual. This approach has the benefit of being effective and relatively simple, both in terms of its implementation and mathematical analysis. The price of this simplicity is that this method can not be used efficiently with generalized eigenvalue problems, and it precludes the efficient use of most preconditioners for solving the linear systems.

The other method is a somewhat more substantial modification of the FEAST algorithm: the contour integral of standard FEAST is modified in such a way that the solution of the linear systems of equations requires a roughly constant amount of work at each FEAST iteration. This more substantial modification can be used to solve both standard and generalized eigenvalue problems efficiently, and it allows for the efficient use of any preconditioners in solving the linear systems of equations. It will also be the key to using FEAST for the solution of nonlinear eigenvalue problems (which will be the subject of Chapter 5).

Much as the standard FEAST algorithm is a generalization of Shift and Invert Iteration that uses multiple shifts, these modifications correspond to multishift generalizations of Inexact Shift and Invert Iterations, and Generalized Inexact Shift and Invert Iterations (see Section 2.3.6 on page 45).

4.2 Basic IFEAST

The basic IFEAST algorithm is described in Algorithm 7. The only substantial difference between the IFEAST algorithm as presented in Algorithm 7 and the standard FEAST algorithm as presented in Algorithm 6 (on page 65) is that the linear systems of equations in IFEAST are deliberately solved inaccurately. Specifically, they are solved such that the residual norm for each linear system right hand side is below some fraction α of the maximum eigenvector residual, i.e.

$$||BX^{(i)}e_j - \frac{1}{\omega_k}(z_kB - A)Y_k^{(i)}e_j|| \leq \alpha \max_{1\leq l\leq m_0} ||\lambda_l Bx_l - Ax_l||, \quad \lambda_l \text{ inside } \mathcal{C} \qquad (4.5)$$

Much as the standard FEAST algorithm is a generalization of Shift and Invert Iteration that uses multiple shifts in the complex plane, IFEAST as described in Algorithm 7 is a generalization of Inexact Shift and Invert Iteration (see Section 2.3.6 on page 45) that uses multiple shifts in the complex plane.

The IFEAST algorithm still converges linearly to the eigenvectors of interest, despite the fact that the linear systems of equations are solved only approximately. Moreover, for standard eigenvalue problems (i.e. $B = I$), the number of iterations that are needed for a Krylov subspace-based linear system algorithm to converge to the residual tolerance required by IFEAST is roughly constant at each iteration, despite the fact that this tolerance decreases at each iteration in proportion to the eigenvector residual. The following subsection discusses these observations in more detail, including their implications for the use of preconditioners and for the efficient solution of generalized problems.

Algorithm 7 The Basic IFEAST algorithm for solving $AX = BX\Lambda$

Inputs:

- Matrices[a] $A, B \in \mathbb{C}^{n \times n}$
- Closed contour $\mathcal{C}$ that encloses the search region for eigenvalues in the complex plane
- Overestimate m_0 for the number of eigenvalues inside $\mathcal{C}$
- Initial guess $\tilde{X}^{(0)} \in \mathbb{C}^{n \times m_0}$ for the search subspace spanned by the solution to the eigenvalue problem
- Set of n_c quadrature weights and points (ω_k, z_k) for numerically integrating equation (4.1)
- Initial value for FEAST eigenvector residual $||R_F||$ [b]
- Relative tolerance α for linear system residuals, with $0 < \alpha < 1$

For each subspace iteration i:

1. Iteratively solve n_c shifted linear systems, each with m_0 right hand sides, for $Y_k^{(i)} \in \mathbb{C}^{n \times m_0}$.

$$\frac{1}{\omega_k}(z_k B - A)Y_k^{(i)} = B\tilde{X}^{(i)}, \quad 1 \leq k \leq n_c \tag{4.4}$$

such that the iterations are stopped when the following tolerance on the linear system residuals is met:

$$||B\tilde{X}^{(i)}e_j - \frac{1}{\omega_k}(z_k B - A)Y_k^{(i)}e_j|| \leq \alpha||R_F|| \quad \forall j,\ 1 \leq j \leq m_0$$

where e_j is the j^{th} canonical unit vector (i.e. the above tolerance must be met for each of the individual right hand sides).

2. Form the filtered subspace Q

$$Q = \rho(A, B)\tilde{X}^{(i)} = \sum_{k=1}^{n_c} Y_k^{(i)}$$

3. Perform Rayleigh-Ritz procedure to find a new estimate for eigenvalues and eigenvectors:

i. Solve reduced eigenvalue problem for $X_Q \in \mathbb{C}^{m_0 \times m_0}$

$$A_Q X_Q = B_Q X_Q \tilde{\Lambda}$$

with $A_Q = Q^H A Q$ and $B_Q = Q^H B Q$

ii. Get new estimate for subspace $\tilde{X}^{(i+1)}$: $\tilde{X}^{(i+1)} = QX_Q$

4. Calculate the FEAST eigenvector residual $||R_F|| = max\ ||\tilde{\lambda}_j B\tilde{x}_j^{(i+1)} - A\tilde{x}_j^{(i+1)}||,\ 1 \leq j \leq m_0,\ \lambda_j$ inside $\mathcal{C}$. If R_F is above a given tolerance, **GOTO 1**.

Outputs: Diagonal matrix $\tilde{\Lambda}$ of approximations for the eigenvalues inside $\mathcal{C}$ and corresponding approximate eigenvectors $\tilde{X}$.

---

[a] Although this algorithm *can* be used for eigenvalue problems where $B \neq I$, it will be very inefficient; when $B \neq I$ the solution of Equation (4.24) requires an increasing number of iterations as the eigenvalue problem converges.

[b] This can be calculated exactly, but I find a good initial value to simply be $||R_F||$=1

4.2.1 Convergence

The standard FEAST algorithm is a variation of Subspace Iteration, and so it can be analyzed quantitatively in the same way. The inequality

$$||\tilde{q}_j - x_j|| \leq \left(\frac{\gamma_{m_0+1}}{\gamma_j}\right) ||q_j - x_j||, \tag{4.6}$$

provides an upper bound on the error of approximation for the eigenvector x_j in the subspace spanned by Q after doing a single FEAST iteration. Because the upper bound on the eigenvector error after doing a single FEAST iteration (i.e. $||\tilde{q}_j - x_j||$) is proportional to an upper bound on the eigenvector error before doing that FEAST iteration (i.e. $||q_j - x_j||$), FEAST is said to converge linearly. Section 3.3 (on page 66) discusses this inequality and its implications in more detail.

The Inexact FEAST algorithm is not a variation of Subspace Iteration; it is, instead, an approximation of Subspace Iteration[1]. A corresponding inequality for eigenvector errors with IFEAST is

$$||\tilde{q}_j - x_j|| \leq \left(\frac{\gamma_{m_0+1} + \alpha_j\Delta}{\gamma_j}\right) ||q_j - x_j||. \tag{4.7}$$

As in equation (4.6), $||q_j - x_j||$ is the error in the estimation of eigenvector j at the current IFEAST iteration, $||\tilde{q}_j - x_j||$ is an upper bound on the eigenvector error for eigenvector j at the next IFEAST iteration, and γ_k is the k^{th} largest magnitude eigenvalue of the operator $\rho(A, B)$ in equation (4.1), where the linear systems of FEAST are solved exactly.

Equation (4.7) contains two additional terms that account for the fact that the FEAST linear systems are solved inexactly. One of these terms is α, which is a

[1]Because the linear systems of equations are solved approximately at each IFEAST iteration, it is implicitly implementing Subspace Iteration such that the operator for performing matrix vector products changes at each iteration.

measure of how accurately the linear systems of FEAST are solved relative to the current eigenvector error $||q_j - x_j||$. If the FEAST linear system right hand sides from equation (4.24) are solved such that

$$||BXe_j - \frac{1}{\omega_k}(z_kB - A)\tilde{Y}e_j|| \leq \epsilon, \quad 1 \leq j \leq m_0 \tag{4.8}$$

is the tolerance on the linear system residuals, then α_j is defined as the ratio of the magnitude of the maximum FEAST linear system residual ϵ to the value of the eigenvector error for the j^{th} eigenvector, i.e.

$$\alpha_j = \epsilon / ||q_j - x_j||. \tag{4.9}$$

The other term, Δ, is defined as

$$\Delta = \sum_{k=1}^{n_c} ||\omega_k(z_kB - A)^{-1}||. \tag{4.10}$$

When $B = I$ then Δ is a measure of how close the FEAST quadrature points z_k are to the eigenvalues of A. In that case, the farther the z_k are from the eigenvalues of A in the complex plane, the smaller Δ becomes. When $B \neq I$ then the interpretation of Δ is less straight forward; in that case Δ is bounded such that

$$\Delta \leq ||B|| \sum_{k=1}^{n_c} ||\omega_k(z_kI - B^{-1}A)^{-1}||, \tag{4.11}$$

meaning that its value is related to both the distance of the eigenvalues of the matrix pencil (A, B) from the shifts z_k and to the spectrum of the matrix B.

The details of the derivation of equation (4.7) are given in Appendix A.

Whether or not IFEAST converges to the eigenvectors of interest is determined by the coefficient for the value of $||q_j - x_j||$ on the right hand side of equation (4.7); if this coefficient is less than one, i.e.

$$\left(\frac{\gamma_{m_0+1} + \alpha_j \Delta}{\gamma_j}\right) < 1, \tag{4.12}$$

then the eigenvector error at the next iteration is guaranteed to be smaller than the eigenvector error at the current iteration.

The condition for IFEAST to converge is then

$$\alpha_j \Delta < |\gamma_j| - |\gamma_{m_0+1}|. \tag{4.13}$$

If the linear systems of equations are solved sufficiently accurately then α_j will be very small, and inequality (4.13) will be satisfied. Exactly how accurately the linear systems of equations need to be solved in order to ensure convergence depends on the discretization of the contour integral and on the spectrum of the matrices whose eigenvalue problem is being solved. It is generally not possible to know the most pertinent information about these things beforehand - the eigenvalue problem is what we are trying to solve, after all - and so the decision about how accurately to solve FEAST's linear systems of equations requires problem-dependent heuristic estimation.

Part of this heuristic estimation, for the version of IFEAST in Algorithm 7, is to replace the eigenvector error $||q_j - x_j||$ with the eigenvector residual $||\tilde{\lambda} B\tilde{x} - A\tilde{x}||$ when determining the linear system residual tolerance. This heuristic, or something like it, is necessary because the eigenvector error can not be known before the actual eigenvector solution is calculated.

If the linear systems of equations (4.4) from Algorithm 7 are solved exactly, then $\alpha_j = 0$ for all j and we recover the convergence rate of standard FEAST. If the linear

systems of equations (4.24) are solved inexactly, then IFEAST will converge more slowly[2] than standard FEAST will for the same problem and set of parameters m_0 and n_c. The more inaccurately the linear systems are solved, the more slowly IFEAST converges, and the degree to which the accuracy of the linear system solves affects the rate of convergence depends on the spectrum of (A, B), through both the values of γ_j and Δ. If the right hand sides of the linear systems of equations (4.24) are solved to the same level of accuracy relative to the error in the estimation of the eigenvector subspace at each iteration, then IFEAST converges linearly just as standard FEAST does.

4.2.2 Computational Efficiency

Inequalities (4.8) and (4.9) tell us that, in order to have IFEAST converge linearly, as standard FEAST does, the linear systems of IFEAST must be solved more accurately than the current eigenvector error, and so the tolerance on the linear system residuals decreases as the approximate eigenvectors converge to the actual eigenvector solutions. This would seem to imply that the number of matrix-vector multiplications must also increase as the IFEAST algorithm converges, which does not bode well for performance.

In fact, much as with Inexact Shift and Invert iteration (of which IFEAST is a generalization), this is not the case when $B = I$. In general, when using Krylov subspace-based iterative solvers for solving the linear systems in IFEAST, the number of matrix vector products that needs to be performed at each iteration of IFEAST is approximately constant. This occurs with IFEAST for the same essential reason that it occurs with Inexact Shift and Invert Iteration: the closer the right hand side of a linear system is to being an invariant subspace of its coefficient matrix, the easier

[2]More slowly in terms of the reduction in the eigenvector error per FEAST iteration, anyway; the total time to solution is proportional to the number of FEAST iterations multiplied by the number of linear system iterations at each FEAST iteration.

that linear system of equations is to solve. The right hand sides of the IFEAST linear systems $(z_k I - A)y = x$ converge to invariant subspaces of the matrices $(z_k I - A)$ at the same time as the tolerance on the residual for their solution is decreased, with the result that the difficulty of their solution remains roughly constant for each iteration of IFEAST. Appendix C discusses this effect more quantitatively and rigorously, and references [64] and [19] discuss this effect in more detail in the context of Inexact Shift and Invert Iteration.

Because we can expect the number of linear system iterations to be constant at each IFEAST iteration, we can also consider a different heuristic as an alternative to trying to guess a value of α that satisfies the convergence condition in Equation (4.13). Instead, we can guess the number of linear system iterations that will be required for the convergence condition in Equation (4.13) to be satisfied, and simply use that same number of linear system iterations at each IFEAST iteration. This is the approach that I will use in all of the examples in this dissertation, in part because of its simplicity, and in part because it will help to clearly illuminate the relationship between IFEAST and polynomial filter functions (which we will discuss in Section 4.4 on page 85).

IFEAST is an efficient means of solving standard eigenvalue problems because its linear systems of equations can be solved by using a roughly constant number of linear system iterations at each IFEAST iteration. This effect disappears when the right hand sides of the linear systems of equations do not converge to invariant subspaces of the coefficient matrix, which is the case for generalized eigenvalue problems and when using preconditioners to solve the linear systems of equations.

For generalized eigenvalue problems, the FEAST linear systems of equations are

$$(z_k B - A)Y_k = BX. \tag{4.14}$$

Even when X has converged to the exact eigenvectors of $AX = BX\Lambda$, BX is not an invariant subspace of $(z_k B - A)$. For a left or right preconditioner P, the FEAST linear systems of equations are

$$(z_k I - A)P^{-1}Y_k^{(P)} = X \quad \text{or} \quad P^{-1}(z_k I - A)Y_k = P^{-1}X, \tag{4.15}$$

where $Y_k^{(P)} = PY_k$. In either case the right hand sides are generally not invariant subspaces of the coefficient matrix.

In both the case of generalized eigenvalue problems and the case of preconditioned matrices, the number of required linear system iterations increases as the approximate eigenvector solutions converge. This is especially ironic for the use of preconditioners, which are supposed to make the solution of linear systems of equations easier rather than more difficult. This problem can be solved with the use of "tuned" preconditioners, which are a particular modification of standard preconditioners that preserves the constant iteration-number properties of linear system solvers for Inexact Shift and Invert Iteration [17, 20, 64]. Tuned preconditioners can also be used to help recover efficiency for solving linear systems of equations with Inexact Shift and Invert Iteration for the generalized eigenvalue problem [107]. I do not explore these options in this dissertation, however, and instead use a different approach that is both highly effective and which can be extended to a larger number of problem domains.

The following section describes this other approach, which is a version of IFEAST that uses a different form of the contour integral. This alternative contour integral allows the linear systems of equations to be solved with a constant number of iterations even for preconditioned matrices and generalized eigenvalue problems.

4.3 IFEAST for Generalized Eigenvalue Problems

IFEAST for the generalized eigenvalue problem uses the contour integral

$$Q = \frac{1}{2\pi i} \oint_{\mathcal{C}} \left(\tilde{X} - (zB - A)^{-1}(B\tilde{X}\tilde{\Lambda} - A\tilde{X}) \right) (zI - \tilde{\Lambda})^{-1} dz \tag{4.16}$$
$$= \frac{1}{2\pi i} \oint_{\mathcal{C}} \left(\tilde{X} - Y(z) \right) (zI - \tilde{\Lambda})^{-1} dz$$

where $\tilde{X}$ is a matrix of Ritz vectors and $\tilde{\Lambda}$ is a diagonal matrix of the corresponding Ritz values, rather than the standard FEAST contour integral

$$Q = \frac{1}{2\pi i} \oint_{\mathcal{C}} (zB - A)^{-1} B\tilde{X} dz. \tag{4.17}$$

The benefit of using the contour integral in Equation (4.16) is that its evaluation requires the solution of linear systems of equations of the form

$$(zB - A)Y(z) = R_E, \tag{4.18}$$

where $R_E = B\tilde{X}\tilde{\Lambda} - A\tilde{X}$ is a block matrix of eigenvector residuals.

When the linear systems of equations (4.18) are solved exactly, for example by using factorization-based numerical methods, then the integrals in Equations (4.16) and (4.17) are exactly equal to each other. The details of the derivation of Equation (4.16) from Equation (4.17) are described in Appendix B, and another (simpler, but less rigorous) derivation is provided in Chapter 5 in the context of nonlinear eigenvalue problems.

If the linear systems of equations (4.18) are solved inexactly with tolerance ϵ on their relative residuals, then

$$\epsilon < \alpha |z - \tilde{\lambda}_i| \tag{4.19}$$

is the condition on the value of ϵ that guarantees that

$$||B\tilde{x}_i - (zB - A)\tilde{y}_i(z)|| < \alpha ||\tilde{\lambda}_i B\tilde{x}_i - A\tilde{x}_i||, \tag{4.20}$$

where $\tilde{x}_i$ and $\tilde{\lambda}_i$ are the i^{th} Ritz vector and value, and $\tilde{y}_i(z)$ is the i^{th} approximate linear system solution. In other words, once the eigenvalue solutions have converged enough that the difference $z - \tilde{\lambda}_i$ does not change much in between iterations[3], solving the linear systems of equations in Equation (4.18) to a *constant* tolerance is sufficient to ensure that the original IFEAST heuristic is satisfied and that the norms of the residuals for the original FEAST linear systems of equations are below some fraction α of the eigenvector residual norm. This is true regardless of whether or not a preconditioner is used, and regardless of whether the eigenvalue problem is standard or generalized.

The reason that this works is because it takes advantage of the spectral information that is contained in the approximate eigenvectors and eigenvalues that have been calculated by IFEAST. Equation (4.18) is the correction equation that results from applying a specific variation of iterative refinement (i.e. using a particular kind of linear system restart) to solving the standard FEAST linear system

$$(zB - A)Y(z) = B\tilde{X} \tag{4.21}$$

with the initial guess

$$\tilde{Y}(z) = \tilde{X}(zI - \tilde{\Lambda})^{-1}, \tag{4.22}$$

which is the exact solution in the case that $\tilde{X}$ and $\tilde{\Lambda}$ are the exact eigenvectors and eigenvalues. When they are not the exact eigenpairs, the linear system residual for the initial guess $\tilde{Y}(z)$ is proportional to the eigenvector residual for the approximate eigenpairs $\tilde{X}$ and $\tilde{\Lambda}$, and one can essentially reuse the work that was done for solving previous linear systems of equations by solving the correction equation in Equation

[3]The approximate eigenvalues $\tilde{\lambda}_i$ do not have to be highly accurate for this to happen; being accurate even to a single significant digit is often enough to ensure that the $z - \tilde{\lambda}_i$ is essentially constant.

(4.18). A detailed explanation and derivation of Equation (4.19) is provided in the Appendix in Section B.3 on page 183.

The generalized IFEAST algorithm is the result of implementing the FEAST algorithm with the integral in Equation (4.16) and solving the associated linear systems of equations to a constant tolerance α on the norms of the linear system residuals. Similarly to the previous variation of IFEAST, Generalized IFEAST is a generalization of Generalized Inexact Shift and Invert iteration that uses multiple shifts in the complex plane. Generalized IFEAST is described in Algorithm 8. This (with or without solving the linear systems of equations inexactly) is the recommended variation of the FEAST algorithm, and it will form the basis of the future 4.0 release of the FEAST software package.

4.4 IFEAST as a Polynomial Filter Algorithm

The standard FEAST algorithm can be interpreted as a method for implementing a rational function filter in order to selectively calculate the eigenvectors of a matrix pencil whose eigenvalues lie in a particular, user-defined region of the complex plane. All of the behavior of the FEAST algorithm can be explained in terms of the action of this rational function filter on the coordinates of an approximate eigenvector.

For standardized eigenvalue problems the IFEAST algorithm has a similar interpretation. Whereas the FEAST algorithm implements a rational function filter, IFEAST implements a polynomial function filter. This is necessarily true; IFEAST uses only matrix-vector multiplication in order to refine an approximate eigenvector subspace, and so the approximate eigenvector subspace at one iteration is necessarily the result of applying some matrix polynomial to the eigenvector subspace from the previous iteration. The particular polynomial function that IFEAST applies to an approximate subspace depends on the linear system solving algorithm that is used and on the approximate subspace itself.

Algorithm 8 The General IFEAST algorithm for solving $AX = BX\Lambda$

Inputs:

- Matrices $A, B \in \mathbb{C}^{n\times n}$
- Closed contour $\mathcal{C}$ that encloses the search region for eigenvalues in the complex plane
- Overestimate m_0 for the number of eigenvalues inside $\mathcal{C}$
- Initial guess $\tilde{X}^{(0)} \in \mathbb{C}^{n\times m_0}$ for the search subspace spanned by the solution to the eigenvalue problem
- Set of n_c quadrature weights and points (ω_k, z_k) for numerically integrating equation (4.1)
- Tolerance α for linear system relative residuals, with $0 < \alpha < 1$

For each subspace iteration i:

0. Set $Q = \tilde{X}^{(0)}$

1. Perform Rayleigh-Ritz procedure to find a new estimate for eigenvalues and eigenvectors:

i. Solve reduced eigenvalue problem for $X_Q \in \mathbb{C}^{m_0\times m_0}$

$$A_Q X_Q = B_Q X_Q \tilde{\Lambda}$$

with $A_Q = Q^H A Q$ and $B_Q = Q^H B Q$

ii. Get new estimate for subspace $\tilde{X}^{(i)}$: $\tilde{X}^{(i)} = Q X_Q$

2. Calculate the FEAST eigenvector residuals $R_E = B\tilde{X}^{(i)}\tilde{\Lambda} - A\tilde{X}^{(i)}$. If

$$\max_{1\le j\le m_0} ||A\tilde{x}_j^{(i)} - \tilde{\lambda}_j B\tilde{x}_j^{(i)}||, \quad \tilde{\lambda}_j \text{ inside } \mathcal{C} \tag{4.23}$$

is below a given tolerance, **EXIT**.

3. Iteratively solve n_c shifted linear systems, each with m_0 right hand sides, for $Y_k^{(i)} \in \mathbb{C}^{n\times m_0}$.

$$\frac{1}{\omega_k}(z_k B - A)Y_k^{(i)} = R_E, \quad 1 \le k \le n_c \tag{4.24}$$

such that the iterations are stopped when the following tolerance on the linear system residuals is met:

$$||R_E e_j - \frac{1}{\omega_k}(z_k B - A)Y_k^{(i)} e_j|| \le \alpha \ \ \forall j,\ 1 \le j \le m_0$$

where e_j is the j^{th} canonical unit vector (i.e. the above tolerance must be met for each of the individual right hand sides).

4. Form the filtered subspace Q

$$Q = \sum_{k=1}^{n_c} \omega_k \left(\tilde{X}^{(i)} - Y_k^{(i)}\right)(z_k I - \tilde{\Lambda})^{-1}$$

5. **GOTO** Step 1.

Outputs: diagonal matrix $\tilde{\Lambda}$ of approximations for the m eigenvalues inside $\mathcal{C}$, and approximations for the corresponding eigenvectors $\tilde{X}$.

Consider a single approximate eigenvector $\tilde{x}$, which we filter using the usual FEAST integral

$$q = \frac{1}{2\pi i} \oint (zI - A)^{-1}\tilde{x} \; dz \tag{4.25}$$

If the integrand $(zI - A)^{-1}\tilde{x}$ is calculated approximately by using k iterations of any linear system solver (with no restarts), then the right hand side of Equation (4.25) becomes

$$\frac{1}{2\pi i} \oint p(zI - A)\tilde{x} \; dz, \tag{4.26}$$

where $p(zI - A)$ is the degree k matrix polynomial

$$p(zI - A) = \sum_{i=0}^{k} a_i (zI - A)^i. \tag{4.27}$$

The total polynomial filter that IFEAST applies to $\tilde{x}$ is thus

$$\rho_p(A) = \frac{1}{2\pi i} \oint p(zI - A) \; dz. \tag{4.28}$$

The following subsections describe the properties of $p(zI - A)$, and how these explain the behavior of the total filter $\rho_p(A)$.

4.4.1 IFEAST Polynomial Filters from GMRES

In the case of GMRES it is relatively straight-forward to explicitly describe the properties of the matrix polynomial $p(zI - A)$, and therefore to draw conclusions about $\rho_p(A)$. GMRES calculates the matrix-vector product $p(zI - A)\tilde{x}$ by (implicitly) finding the polynomial that solves the minimization problem

$$\min_p ||\tilde{x} - (zI - A)p(zI - A)\tilde{x}||^2, \tag{4.29}$$

where $||\tilde{x} - (zI - A)p(zI - A)\tilde{x}||^2$ is the squared residual norm of the original linear system of equations. This squared residual norm can also be written as

$$||(zI - A)\left((zI - A)^{-1}\tilde{x} - p(zI - A)\tilde{x}\right)||^2. \tag{4.30}$$

Thus, using k iterations of GMRES with $\tilde{x}$ as the right hand side is similar to (but not exactly the same as)[4] finding the degree k matrix polynomial $p(zI - A)$ that best approximates the action of the matrix inverse $(zI - A)^{-1}$ on the vector $\tilde{x}$. If we write the approximate eigenvector $\tilde{x}$ (which may, in fact, be totally random) in the basis of components of the exact eigenvectors x_i (which I hereafter call an "eigenbasis"), i.e.

$$\tilde{x} = \sum_{j=1}^{n} c_j x_j, \tag{4.31}$$

then Equation (4.29) becomes

$$\min_p \left\| \sum_{j=i}^{n} c_j (z - \lambda_j)\left((z - \lambda_j)^{-1} - p(z - \lambda_j)\right) x_j \right\|^2. \tag{4.32}$$

Writing out the norm in Equation (4.32) explicitly, we get

$$\min_p \sum_{i,j=1}^{n} \left((z - \lambda_i)^{-1} - p(z - \lambda_i)\right)^* c_i^* c_j (z - \lambda_i)^* (z - \lambda_j) x_i^H x_j \left((z - \lambda_j)^{-1} - p(z - \lambda_j)\right) \tag{4.33}$$

Solving Equation (4.33) with respect to the polynomial p, as GMRES does, is a generalized least squares problem. Using Equation (4.27) for the definition of p [5], we

---

[4]This is not *exactly* the same as finding the best polynomial $p(zI - A)$ to approximate the action of the matrix inverse $(zI - A)^{-1}$ because the norm is weighted by the matrix product with $zI - A$ on the left of Equation (4.30).

[5]The polynomial p can be written in any polynomial basis set, e.g. Legendre or Chebyshev polynomials; I use Equation (4.27) for the sake of simplicity.

can see this by defining the following matrices and vectors by their elements:

$$b_i = (z - \lambda_i)^{-1}, \quad 1 \leq i \leq n \tag{4.34}$$

$$M_{il} = (z - \lambda_i)^l, \quad 1 \leq i \leq n,\ 0 \leq l \leq k \tag{4.35}$$

$$W_{ij} = c_i^* c_j (z - \lambda_i)^* (z - \lambda_j) x_i^H x_j, \quad 1 \leq i \leq n,\ 1 \leq j \leq n \tag{4.36}$$

Equation (4.33) can then be written in terms of a quadratic form as

$$\min_a (b - Ma)^H W (b - Ma), \tag{4.37}$$

where a is the vector of coefficients a_i that defines the polynomial p, as in Equation (4.27). Equation (4.37) is a generalized least squares problem[6], the solution to which is

$$a = \operatorname*{argmin}_a (b - Ma)^H W (b - Ma) = (M^H W M)^{-1} M^H W b \tag{4.38}$$

The following subsection discusses the properties of $p(zI - A)$ and $\rho_p(A)$ in terms of the solutions to the minimization problem in Equation (4.37) for eigenvalue problems with normal matrices, which makes the analysis especially clear. Although this analysis applies most directly to GMRES, which finds the actual solution of Equation (4.37), we find that our conclusions also apply qualitatively to other Krylov-based linear system solvers as well, since they still calculate approximate solutions to it. Figure 4.6 (on page 110) shows examples of an IFEAST polynomial filter that has been calculated by using several different linear system solving algorithms.

---

[6]Generalized least squares is best-known in the context of linear regression, where it is used to fit linear statistical models in which the noise components are correlated rather than independent [42]. When W is diagonal then it is also known as "weighted least squares".

4.4.2 Normal Matrices

If the matrix A is normal, then Equation (4.33) becomes

$$\sum_{i=1}^{n} |c_i|^2 |z - \lambda_i|^2 \left| (z - \lambda_i)^{-1} - p(z - \lambda_i) \right|^2, \tag{4.39}$$

and W (from Equation (4.37)) is a diagonal matrix with diagonal entries $|c_i|^2|z - \lambda_i|^2$. If $W = I$ then GMRES calculates the k-degree polynomial that best fits (in a conventional least squares sense) the function $(z - \lambda)^{-1}$ when it is sampled only at the eigenvalues λ_i of A.

The fact that the polynomial $p(z - \lambda)$ is only fitted at values of λ that are eigenvalues of A is a key feature of the IFEAST polynomial filters. It ensures that computation is spent only on fitting the IFEAST polynomial filter in the places where it will actually be evaluated. When the polynomial degree is $k = n - 1$, where n is the dimension of the eigenvalue problem, then the polynomial fit is perfect and IFEAST polynomial filter is exactly equal to the standard FEAST rational filter at the eigenvalues of A. For lower-degree polynomials the fit of the IFEAST filter is less accurate. This is illustrated in Figure 4.1 (on page 102).

The case $W = I$ will never occur in practice; it only happens when the A eigenbasis components of the approximate eigenvector $\tilde{x}$ are such that

$$|c_i| = \frac{1}{|z - \lambda_i|}, \tag{4.40}$$

and that can only be true for one of the n_c different quadrature points z. A more realistic scenario is that c_i is some constant value c for all values of i; we would expect this to be approximately true when $\tilde{x}$ is a randomly-chosen initial guess for the IFEAST algorithm, in which case all of the components of $\tilde{x}$ in the eigenbasis of A are roughly equally likely to be given weight. In this case the values of the diagonal elements of W are determined solely by the squared distances of the shift z from the

locations of the eigenvalues λ_i in the complex plane. W here serves as a weighting matrix for the least squares problem: we are again calculating a k-degree polynomial to fit the function $(z-\lambda)^{-1}$ when it is sampled at the eigenvalues of A, but this time the larger-magnitude values of $z-\lambda_i$ are given greater weight in the minimization problem.

This at first may seem to run contrary to our goal, which is to calculate the eigenvalues that are inside the contour on which z lies; instead, GMRES explicitly tries to make a good fit to $(z-\lambda_i)^{-1}$ for values of λ_i that are far away from the contour. This is actually a desirable feature when considering the final IFEAST polynomial filter, however. If one were to build a polynomial filter explicitly by, for example, fitting a k-degree polynomial to the indicator function for the region of interest in the complex plane (rather than building a filter implicitly by using contour integration and linear system solves as we do here), then it would be necessary to have estimates for the boundaries of the spectrum of A in order to ensure that the polynomial filter takes smaller values on all of the unwanted eigenvalues in the spectrum [50, 79, 110], especially the extremal ones. Implicitly calculating a polynomial by solving linear systems of equations with GMRES, on the other hand, automatically ensures that the need to fit the polynomial for the eigenvalues of A that are far away from the contour $\mathcal{C}$ is taken into account without first having to estimate the extremal eigenvalues.

The components c_i are usually not uniform. As the approximate eigenvector $\tilde{x}$ converges to some exact eigenvector x_j, the coefficients c_i of $\tilde{x}$ in the A eigenbasis converge towards a delta function $c_i = \delta_{ij}$. If $\tilde{x}$ has converged enough that it has a low eigenvector residual, but is still relatively far from being an exact eigenvector (e.g. if the eigenvector residual is 10^{-4}), then the eigenbasis coefficients c_i will be relatively small for most values of i, and will be large when $c_i = c_j$.

The least squares problem is weighted to be more accurate for eigenvalues λ_i that correspond to larger-magnitude values of c_i; as the approximate eigenvector $\tilde{x}$

converges, then, the function $(z-\lambda)^{-1}$ is fitted more accurately for the eigenvalue λ_j of the eigenvector x_j that $\tilde{x}$ is converging to, and less accurately for all of the other eigenvalues. This suggests another perspective on why the number of linear system iterations that is required at each IFEAST iteration is roughly constant, even for Basic IFEAST: the polynomial filter that IFEAST implicitly uses becomes more accurate for the eigenvalues of interest (and less accurate for all other eigenvalues) as the eigenvector solution converges, even though the degree of the polynomial filter stays the same. This comes entirely from the fact that the polynomial filter is the result of solving a weighted least squares problem in which the weights are determined, in part, by the accuracy of the eigenvector solution. Figure 4.3 (on page 105) illustrates the difference between a polynomial filter that is fitted for an approximate eigenvector that is near convergence, and a polynomial filter that is fitted for an approximate eigenvector that is not.

The polynomial filter that is ultimately applied by IFEAST to the approximate eigenvector $\tilde{x}$ is the quadrature sum of the individual polynomial filters that are calculated by solving linear systems of equations with GMRES, i.e.

$$\rho_p(A) = \sum_{i=1}^{n_c} \omega_i p(z_i I - A) \tag{4.41}$$

$$= \sum_{i=1}^{n_c} \sum_{j=0}^{k} \omega_i a_j(z_i, \tilde{x})(z_i I - A)^j, \tag{4.42}$$

where I have noted explicitly that the polynomial coefficients a_j are functions of both the quadrature point z_i and the approximate eigenvector $\tilde{x}$. The fact that a_j is a function of z_i is especially worth noting because one would normally expect a contour integration of a polynomial (i.e. Equation (4.26)) to be zero, because polynomials have no poles in the complex plane. In fact, these particular polynomials *do* have poles in the complex plane, because the coefficients $a_j(z, x)$ have poles for certain values of z.

The polynomial filter function $\rho_p(A)$ acts directly on the eigenvalues of A in such a way that the matrix vector product $\rho_p(A)\tilde{x}$ brings $\tilde{x}$ closer to the subspace spanned by the eigenvectors of interest. The convergence behavior of the IFEAST algorithm can be analyzed by examining the action of of the polynomial $\rho_p(\lambda)$ on scalar numbers λ; Section 4.4.6 provides several illustrations of this.

4.4.3 Implications for IFEAST Convergence

The convergence rate of standard FEAST for eigenvector j is given by

$$\frac{\rho(\lambda_{m_0+1})}{\rho(\lambda_j)}, \tag{4.43}$$

where the eigenvalues λ_i are numbered such that $\rho(\lambda_i) < \rho(\lambda_{i+1})$. Section 4.2.1 describes an upper bound for IFEAST that contains a similar coefficient which accounts for the errors in the solutions of the linear systems of equations.

We can also analyze the convergence of the IFEAST algorithm by considering it in terms of the application of the polynomial filter in Equation (4.42). The convergence rate of IFEAST might then be

$$\frac{\rho_p(\lambda_{m_0+1})}{\rho_p(\lambda_j)}, \tag{4.44}$$

where ρ_p is the polynomial filter from Equation (4.42), and the eigenvalues λ_i are numbered such that $\rho_p(\lambda_i) < \rho_p(\lambda_{i+1})$.

The convergence rate in Equation (4.44) can generally only offer a qualitative description of the convergence behavior of IFEAST. The reason for this is that the filter function ρ_p depends on the vector to which it is being applied. The IFEAST algorithm applies filters to a set of m_0 basis vectors at every iteration, and the filter that it applies is different for each of those m_0 basis vectors. The examination of any one of those filters by itself is not guaranteed to provide a precise estimate for the convergence rate of IFEAST, and we will see an example in Section 4.6.1 where our

expectations from examining a single filter function are not always matched by the actual behavior of IFEAST.

The fact that value of ρ_p depends on the vector to which it is being applied also means that the filter changes at every iteration, as the approximate eigenvectors of IFEAST converge to exact eigenvectors. As an approximate eigenvector nears convergence, the convergence rate in Equation (4.44) no longer describes the behavior of IFEAST at all. It is based on the standard convergence analysis of Subspace Iteration, which explicitly does not account for the the specific coordinates of the vector to which those iterations are being applied. The filter that is used implicitly by IFEAST depends intimately on the coordinates of the vector to which it is applied; for a vector that accurately approximates an eigenvector, the filter value is fitted very accurately for the eigenvalues of interest, and very poorly everywhere else. Equation (4.44) incorrectly predicts a failure to converge in this case.

Even so, the convergence rate that is predicted by Equation (4.44) for a random or uniformly-distributed initial guess vector does usually explain the convergence behavior of IFEAST, even if it is only a qualitative explanation. This appears to be most true when a large enough number of linear system iterations are used that IFEAST enters and maintains linear convergence.

The most important parameters when analyzing the convergence behavior of standard FEAST are the number of contour integration points n_c and the subspace dimension m_0. Increasing either of these numbers causes standard FEAST to converge more quickly by iteration, and this can be accomplished easily by using additional parallel computation. The same is true of IFEAST, but to a more limited extent. Increasing m_0 always improves the rate of convergence, but the degree to which convergence is improved depends on the quality of the polynomial filter. The quality of the polynomial filter can vary considerably depending on the spectrum of the matrix and on the location of the integration contour in that spectrum.

Increasing n_c does not always increase the convergence rate of IFEAST. From the perspective of polynomial filters, increasing the value of n_c improves the quality of the underlying rational filter to which the polynomial filter is being fitted. If the filter ρ_p is already close to being the best approximation of the desired indicator function that one can fit by using a degree k polynomial, then improving the underlying approximation of the indicator function to which it is being fitted will not improve the quality of the polynomial filter unless the degree k is also increased. In practice, what this means is that the convergence rate of IFEAST will typically improve less with each increase in n_c, until it eventually stops improving at all. The only way to further increase the convergence rate of IFEAST is to use a larger-degree polynomial, which means using more matrix-vector products when solving linear systems of equations, thereby solving them more accurately.

It is worth noting that, although the *convergence rate* of the IFEAST algorithm can be improved to only a limited degree by increasing parallel processing power, this is not necessarily a measure of the actual speed with which IFEAST can solve an eigenvalue problem. We could imagine, for example, that the linear systems of IFEAST might be solved to a very high precision, in which case all of the parallelism of standard FEAST is recovered; IFEAST might none the less take a very large amount of wall-clock time to converge, however, because the number of linear system iterations that would be required for solving the IFEAST linear systems to high precision could be extremely high. We could instead imagine that the linear systems of IFEAST might be solved to a very low precision. In this case the parallelism of IFEAST becomes more limited, but it may still solve an eigenvalue problem very quickly by virtue of the fact that the approximate linear system solves can be performed with only a few linear system iterations. Whether or not IFEAST can be used to solve a particular eigenvalue problem quickly depends fundamentally on how easily the

desired eigenvalues can be filtered by polynomials, which is a limitation that can be mitigated to only a limited degree by using parallel computing of any kind.

4.4.4 Generalized IFEAST

The contour integral for the Generalized IFEAST algorithm differs from that of the Basic IFEAST algorithm, and its interpretation in terms of polynomial eigenvalue filters is somewhat less straight-forward as a result. For Generalized IFEAST the contour integral is

$$q = \frac{1}{2\pi i} \oint \frac{1}{z - \tilde{\lambda}} \Big(I - (zB - A)^{-1}(\tilde{\lambda}B - A) \Big) \tilde{x} \; dz, \tag{4.45}$$

where $\tilde{\lambda}$ is the Ritz value for the approximate eigenvector $\tilde{x}$. As with Basic IFEAST, the matrix inverse product $(zB - A)^{-1}(\tilde{\lambda}B - A)\tilde{x}$ in the integrand is approximated using an iterative solver like GMRES, and Equation (4.45) becomes

$$q = \frac{1}{2\pi i} \oint \frac{1}{z - \tilde{\lambda}} \Big(I - p(zB - A)(\tilde{\lambda}B - A) \Big) \tilde{x} \; dz, \tag{4.46}$$

with $p(zB - A)$ again being a matrix polynomial. Now the polynomial p is the one that solves

$$\min_{p} ||(\tilde{\lambda}B - A)\tilde{x} - (zB - A)p(zB - A)(\tilde{\lambda}B - A)\tilde{x}||^2, \tag{4.47}$$

and the matrix function $\rho_p(A, B)$ that Generalized IFEAST applies to $\tilde{x}$ is

$$\rho_p(A, B) = \sum_{i=1}^{n_c} \frac{\omega_i}{x - \tilde{\lambda}} \Big(I - p(z_i B - A)(\tilde{\lambda}B - A) \Big). \tag{4.48}$$

For the standard eigenvalue problem (i.e. $B = I$), this matrix function becomes

$$\rho_p(A, I) = \sum_{i=1}^{n_c} \frac{\omega_i}{x - \tilde{\lambda}} \Big(I - p(z_i I - A)(\tilde{\lambda}I - A) \Big). \tag{4.49}$$

The matrix function ρ_p in Equation (4.49) is a polynomial filter that acts directly on the eigenvalues of the matrix A. This filter is qualitatively the same as the one that is generated by Basic IFEAST, but the two filters are not exactly equal to each other. Because both filters are produced by solving linear systems of equations using the same kind of minimization problem, they have the same properties with regards to the convergence behavior of IFEAST. Figure 4.6 (on page 110) provides several plots that illustrate the relationship between the polynomial filters for Basic IFEAST and Generalized IFEAST.

In the case of generalized eigenvalue problems (i.e. $B \neq I$) we can no longer interpret ρ_p in Equation 4.48 as a polynomial filter that acts directly on the eigenvalues of the matrix pencil (A, B). The eigenvalue filter interpretation is viable for $B = I$ because

$$p(zI - A) = p(X(zI - \Lambda)X^{-1}) \tag{4.50}$$

$$= Xp(zI - \Lambda)X^{-1}, \tag{4.51}$$

where X is the block matrix of the eigenvectors of A, Λ is the diagonal matrix of the corresponding eigenvalues, and p is any polynomial. When $B \neq I$ then

$$p(zB - A) = p(BX(zI - \Lambda)X^{-1}) \tag{4.52}$$

$$\neq BXp(zI - \Lambda)X^{-1}, \tag{4.53}$$

where X and Λ are now the eigenvectors and eigenvalues of the matrix pencil (A, B).

I have not yet found an intuitive interpretation of the Generalized IFEAST algorithm for generalized problems in terms of polynomial filters. Instead, the generalized eigenvalue problem case appears to have more in common with the nonlinear eigenvalue problem case (see Chapter 5), and they likely have closely-related explanations for their convergence behavior.

4.4.5 Discussions on Polynomial Filters and Jacobi-Davidson

The idea of using polynomial filters to solve eigenvalue problems is already well-established [50, 79, 110]. The typical method for generating a polynomial eigenvalue filter is to choose a filter function that selects the desired eigenvalues, and then calculate a k degree polynomial function that approximates this filter function over the entire convex hull of the eigenvalues of the matrix A.

The IFEAST framework for applying polynomial filters offers some potential benefits over the usual approach. IFEAST does not need to estimate the convex hull of a matrix's spectrum in order to calculate a polynomial filter; it naturally considers the limits of the spectrum by solving the generalized least squares problem. More importantly, IFEAST does not need to fit the filter polynomial over the entire convex hull of the spectrum: it fits the polynomial only at the eigenvalues of the matrix. IFEAST can therefore potentially fit a more accurate filter polynomial, because it does not need to minimize the error in the fit for locations in the complex plane where the filter function will never be evaluated. This especially means that solving nonsymmetric problems, or any eigenvalue problem with complex eigenvalues, will be more tractable with IFEAST because the dimensionality of the set over which it fits the filter polynomial is always the same[7]: zero. The traditional method of fitting a polynomial filter, on the other hand, has to fit the polynomial over a 1D region on the real number line for Hermitian matrices, or a 2D region in the complex plane for nonsymmetric problems. Accurately fitting 2D polynomials, in particular, can require polynomials of much greater degree than accurately fitting polynomials over a 1D region.

---

[7]Even when the eigenvalues are scattered throughout the complex plane, rather than arrayed on the real number line, they still constitute a zero-dimensional set; fitting a polynomial to n points in the complex plane is no more difficult than fitting a polynomial to n points on the real number line.

The polynomial filter perspective also offers a means of relating IFEAST and inexact Simplified Jacobi-Davidson (see Section 2.3.7 on page 51).

With IFEAST the convergence is linear, and the rate of convergence can be increased by improving the contour integration accuracy up until the point that the refinement of the polynomial filter is limited by its (user-determined) maximum degree.

Inexact Simplified Jacobi-Davidson, which is Simplified Jacobi-Davidson with inexact linear system solves, necessarily uses polynomial filters as well. Moreover, much like IFEAST, it implicitly generates a polynomial filter by fitting a polynomial to a rational filter function. The difference between Simplified Jacobi-Davidson and IFEAST is that Simplified Jacobi-Davidson uses only a single shift, whereas IFEAST uses many, and Simplified Jacobi-Davidson moves the shift to be equal to the new Ritz value at each iteration, whereas the IFEAST shifts are constant. We would thus expect that inexact Simplified Jacobi-Davidson would initially converge super-linearly, until the quality of its filter becomes limited by the user-determined maximum degree of the filter polynomial, after which it would be linear. The behavior of inexact Simplified Jacobi-Davidson is then similar to what we would expect to get by increasing the contour integration accuracy of the IFEAST algorithm at each iteration until the rate of convergence no longer improves.

4.4.6 Illustrations

This subsection provides several illustrations of the polynomial filter properties that were discussed in prior subsections by explicitly generating and plotting the polynomial filters for simple example matrices and for simple approximate eigenvectors. For the example matrices I use Hermitian matrices with uniformly-distributed eigenvalues on the real number line; one of these is dimension 12, the other is dimension 1000. For the approximate eigenvectors I use particular, simple linear combinations

of exact eigenvectors. One approximate eigenvector example consists of a sum over all of the exact eigenvectors, i.e.

$$\tilde{x} = \sum_{i=1}^{n} x_i. \tag{4.54}$$

I will refer to this as the "uniformly distributed" approximate eigenvector, because its coefficients in the exact eigenbasis are completely uniformly distributed. It serves as a simple approximation of a random initial guess for an eigenvector[8]. The other approximate eigenvector example is

$$\tilde{x} = \sum_{i=1}^{n} e^{\frac{(\lambda_i - \tilde{\lambda})^2}{4\sigma^2}} x_i, \tag{4.55}$$

where $\tilde{\lambda}$ is a value in the middle of the search contour and $\sigma = 10^{-2}$. I will refer to this as the "near convergence " or the "normally distributed" approximate eigenvector, because the square of its coefficients in the exact eigenbasis approximates a normal distribution. This approximate eigenvector simulates an approximate eigenvector that is closer to convergence, with Ritz value $\tilde{\lambda}$.

4.4.6.1 The Polynomial Filter

Figure 4.1 shows two example polynomial filters that are generated by using a contour integral with GMRES for a 12×12 Hermitian matrix; this is the same matrix and contour integral that was used in the examples in Section 3.4 (on page 68). A degree $n-1$ ($n = 12$ in this case) polynomial exactly fits the values of the polynomial filter to the values of the rational filter at the eigenvalues of the matrix, just as we would expect from a linear least squares fitting of a polynomial to function at a set of n evaluation points.

---

[8] I could, of course, have just as easily used an actual random vector, but this would only serve to produce plots that are less clear, with no added benefit in terms of illustrating the polynomial filter properties that we seek to examine.

Although the exact fitting scenario is implausible in practice - the whole point of using GMRES with FEAST is that sometimes we cannot solve linear systems of equations exactly - it serves to highlight the benefit of using linear system solvers to fit the polynomial, rather than fitting the polynomial to an indicator function over the entire region of interest. Using linear system solvers to fit the polynomial filter allows us to fit it only at the evaluation points of interest (i.e. the eigenvalues of the matrix), which improves the accuracy of the fit at those evaluation points by sacrificing the accuracy of the fit in between them. For the exact fit, the filter polynomial appears to be of relatively poor quality over region of the real number line that we are interested in, but it takes exactly the values that we want it to take at the points where we plan on evaluating it.

Figure 4.1 also shows a degree $n/2$ polynomial filter. In this case the polynomial filter function - even evaluated only at the eigenvalues of the matrix - is of much poorer quality than the original rational filter function that it is being fitted to. This is typical of polynomial filters for matrices of any size, and it provides an illustration of the reason that IFEAST converges more slowly than standard FEAST for inexact linear system solves: the implicit filter function simply reduces the eigenvector error less for IFEAST than it does for standard FEAST.

4.4.6.2 Dependence on the approximate eigenvector

The polynomial filter function depends on the approximate eigenvector to which the IFEAST contour integral is being applied. the examples in Figure 4.1 are polynomial filters for the uniform approximate eigenvector, i.e. Equation (4.54); this approximate eigenvector is similar to what one would get when choosing a random initial guess. Figure 4.2 illustrates this coefficient distribution, as well as the near convergence approximate eigenvector from Equation (4.55), for a different Hermitian

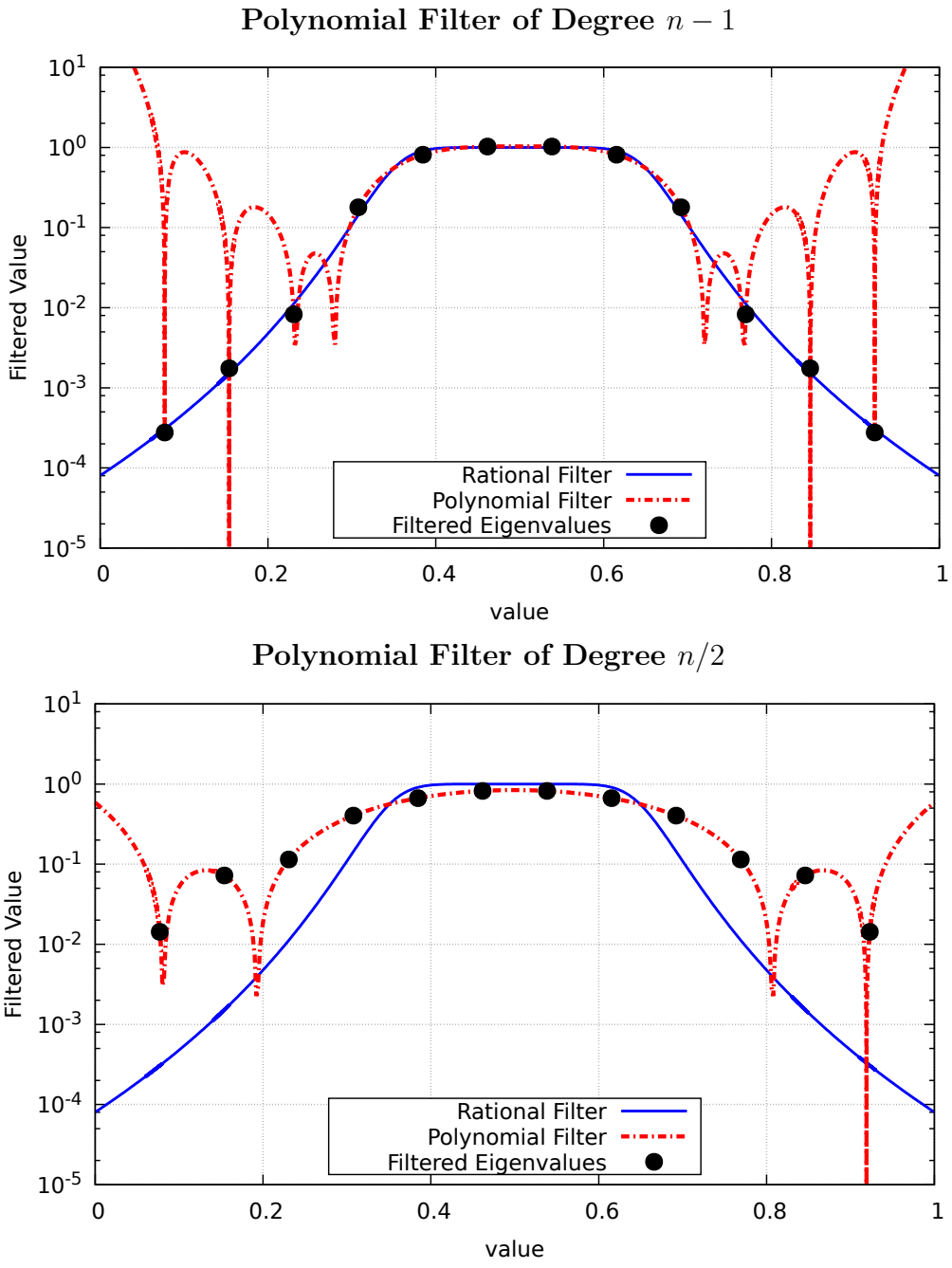


Figure 4.1. *Plots of polynomial filters generated by GMRES for a* 12×12 *Hermitian matrix, along with the rational filter that they are being fitted to. The contour integral for the rational filter selects for the middle 4 eigenvalues and is discretized with 8 quadrature points. These are the same eigenvalues and contour integral that were used in the examples in Section 3.4. The top plot shows a degree 11 polynomial filter (i.e. an exactly-fitted polynomial), and the bottom plot shows a degree 6 polynomial filter.*

matrix of dimension $n = 1000$ with uniformly-distributed eigenvalues over the interval $[1000, 1001]$.

The corresponding polynomial filters for these approximate eigenvectors are plotted in Figure 4.3, along with the original rational filter to which they are being fitted. The filter for the approximate eigenvector with uniformly distributed eigenbasis coefficients is a poor approximation of the original rational filter, but it still is clearly capable of selecting the same eigenvalues as the original rational filter. The filter for the approximate eigenvector that is near convergence is a different story: it is a highly accurate approximation of the original rational filter over a small domain, and it diverges outside of that domain. Figure 4.4 provides a zoomed-in view of all of these filters plotted together, showing that the filter for the approximate eigenvector that is near convergence is fitted extremely well to the original rational filter for the eigenvalues corresponding to high-magnitude eigenbasis coefficients. This is what we would expect based on the original weighted least squares problem, which places more weight on the polynomial fit for eigenbasis coefficients with large magnitudes.

Although the polynomial filter for the near-convergence approximate eigenvector appears to be incapable of successfully selecting for the desired eigenvalues, this is not actually the case: it is only incapable of successfully filtering *most* approximate eigenvectors. When it is applied to the specific approximate eigenvector for which it was fitted, however, the polynomial filter actually will select for the desired eigenvalues. This is illustrated in Figure 4.5, which shows the distributions of eigenbasis coefficients before and after polynomial filtering. For the uniformly-distributed approximate eigenvector, the after-filtering distribution features a peak at the eigenbasis coefficients that correspond to the eigenvalues of interest, indicating successful filtering.

The near-convergence approximate eigenvector, on the other hand, shows two particular effects. The first effect is that the eigenbasis distribution becomes a narrower

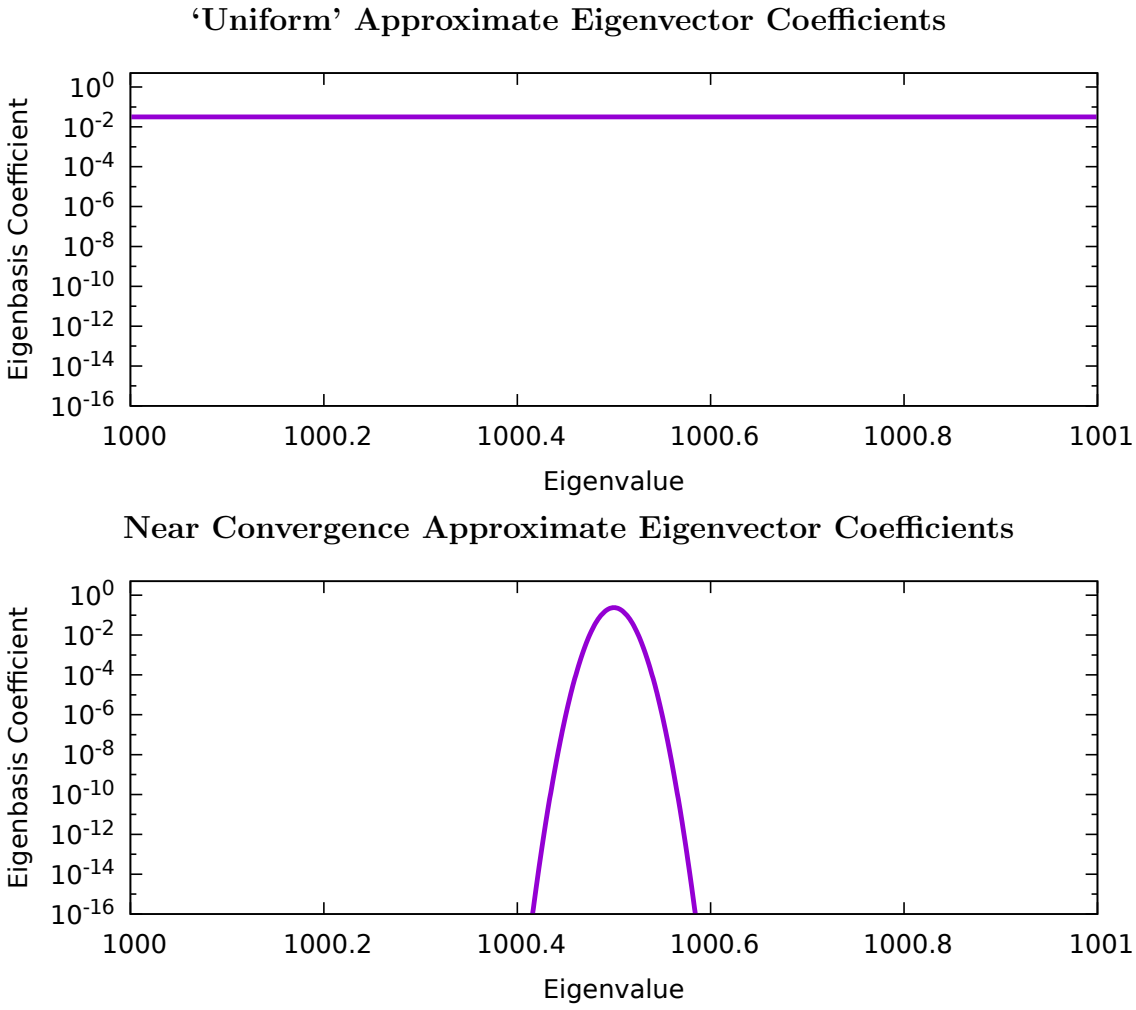


Figure 4.2. *Illustration of two example approximate eigenvectors for a Hermitian matrix of dimension $n = 1000$. Top plot shows an approximate eigenvector whose eigenbasis coefficients are uniformly distributed, making it similar to a random initial guess. Bottom plot shows an approximate eigenvector whose eigenbasis coefficients are normally-distributed, with the mean centered at a particular coefficient whose corresponding eigenvalue is in the middle of the matrix spectrum. Eigen-basis coefficients can be labelled by their corresponding eigenvalue, and so the coefficients in both of these plots are plotted with respect to their corresponding eigenvalue on the x-axis.*

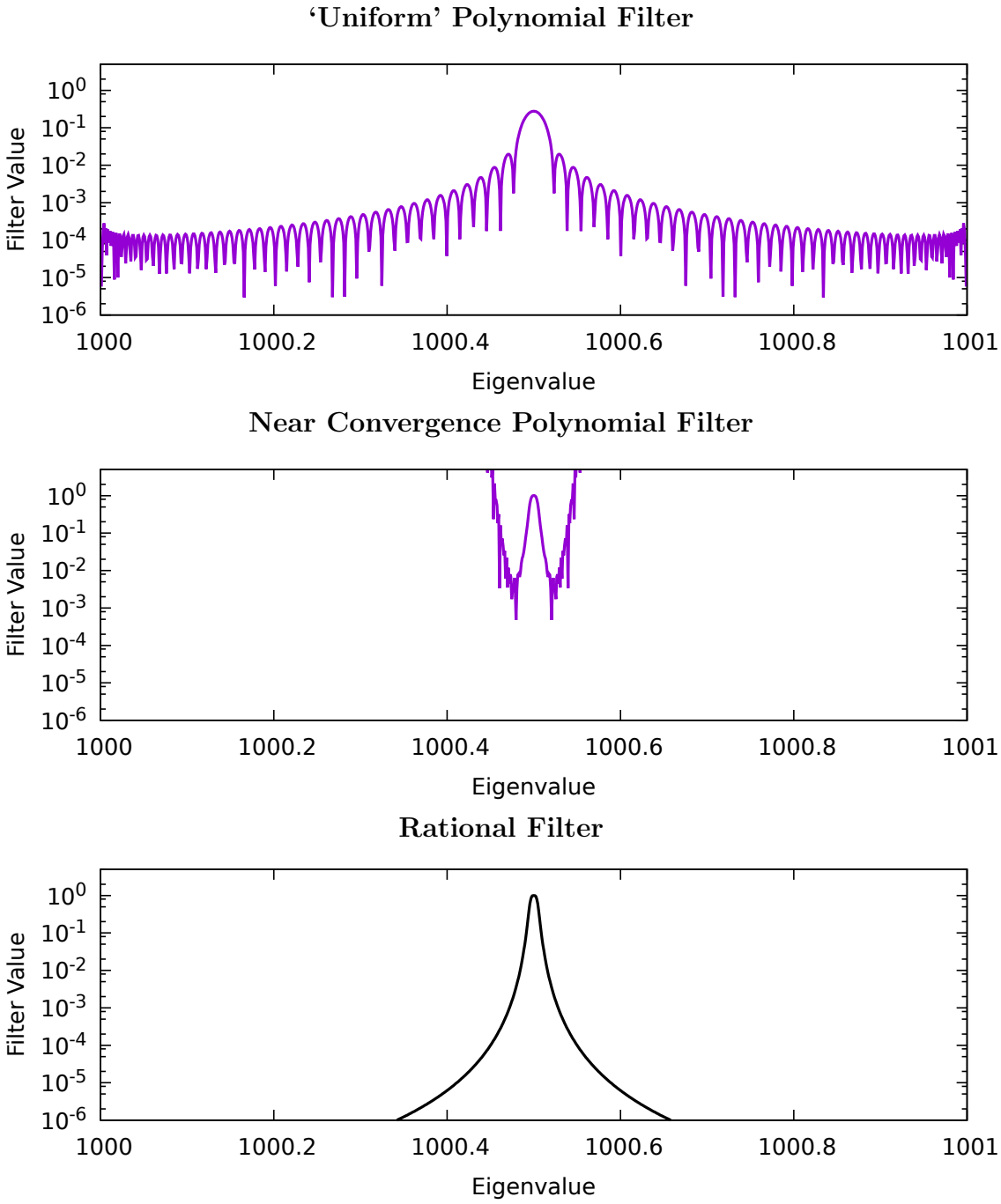


Figure 4.3. *Degree 100 polynomial filters corresponding to the approximate eigenvectors shown in Figure 4.2. Top and middle plots show the polynomial filters for the uniform and near convergence approximate eigenvectors, respectively. Bottom plot shows the original rational filter to which these polynomials are being fitted; this filter selects 10 eigenvalues in the middle of the spectrum.*

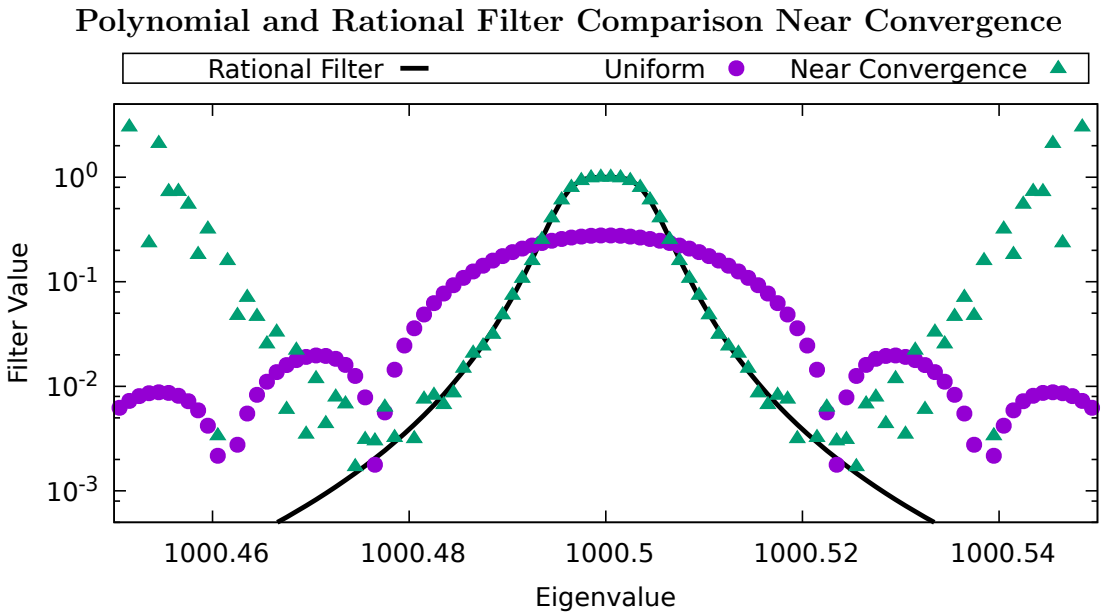


Figure 4.4. *The same filters from Figure 4.3 plotted together. Filter values are plotted for a zoomed-in region of the eigenvalue spectrum. The original rational filter is plotted as a solid line, and the polynomial filters for the two different approximate eigenvectors are shown with plot markers; the polynomial filters are evaluated only at the eigenvalues of the original matrix.*

peak around the eigenbasis coefficients corresponding to the eigenvalues of interest. The other effect is that some of the eigenbasis coefficients that were previously approximately zero are now much larger than zero, becoming approximately equal to 10^{-5}. We can tell, however, that the near-convergence approximate eigenvector is brought closer to convergence through the filtering process by considering its eigenbasis distribution as a probability distribution and calculating its variance; a smaller variance corresponds to a smaller eigenvector residual[9], and indeed the variance for the near-convergence approximate eigenvector in Figure 4.5 goes from being approximately 5×10^{-5} to being approximately 7×10^{-6} after filtering, indicating that it is closer to convergence.

4.4.6.3 Different Linear Solvers

The exact filter that is produced by the IFEAST contour integral depends on the algorithm that is used for solving the linear systems of equations, and on the variation of IFEAST that is used (i.e. Basic or Generalized) . All of the prior results show the filters that are produced by Basic IFEAST using GMRES. In practice, one is much more likely to use algorithms such as MINRES or BiCGSTAB, which can calculate linear system solutions by using short recurrences. Figure 4.6 shows a comparison of the filters that are produced by GMRES, MINRES, and BiCGSTAB when the approximate eigenvector has uniformly-distributed eigenbasis coefficients. Results are shown for both Basic and Generalized IFEAST.

Notably, the filter that is produced by Basic IFEAST with MINRES is slightly different from the filter that is produced by Basic IFEAST with GMRES; they should be the same in exact arithmetic, but numerical error causes them to differ slightly.

[9]This perspective may be familiar from quantum mechanics, in which vectors in a Hilbert space correspond to probability densities for measurement outcomes, which themselves correspond to the eigenvalues of Hermitian operators; the square of the eigenvector residual norm of an approximate eigenvector for a Hermitian matrix is exactly equal to the corresponding variance of the probability density for that matrix's eigenvalues.

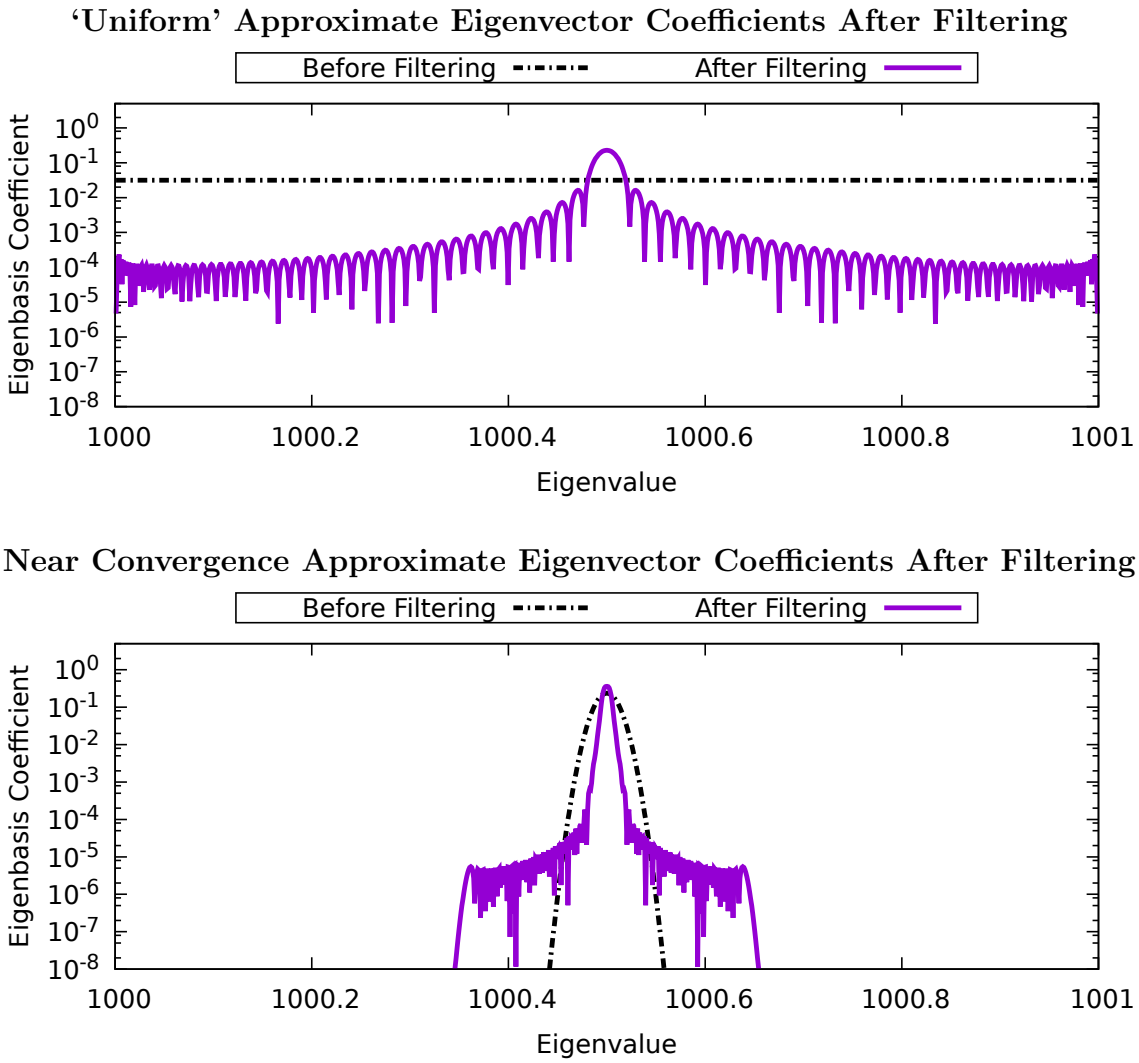


Figure 4.5. *Plots showing the eigenbasis coefficient distributions before and after polynomial filtering. Top plot shows the results for the approximate eigenvector with uniformly distributed coefficients, and the bottom plot shows the results for the approximate eigenvector with normally distributed eigenbasis coefficients (indicating that it is near to convergence).*

Generalized IFEAST with MINRES appears to correct for some of that numerical error.

4.5 Basic IFEAST and Krylov Methods

It is possible, in principle, to implement IFEAST using any kind of linear system solving algorithm whatsoever, provided that the linear systems are solved sufficiently accurately that the IFEAST convergence condition

$$\alpha_j \Delta < |\gamma_j| - |\gamma_{m_0+1}|. \tag{4.56}$$

is met. Although other choices are possible (e.g. the Kaczmarz algorithm [22]), the most likely choice for most practitioners will be to use some sort of Krylov subspace-based algorithm for solving linear systems, since these are known to be powerful and robust.

In light of the fact that there are also Krylov algorithms for solving eigenvalue problems directly, the use of Krylov algorithms for solving the linear systems of equations in IFEAST raises a natural question: if all of the important computation in IFEAST consists of solving linear systems of equations, and all of these linear systems of equations are solved using Krylov subspaces, then is there a relationship between IFEAST and the direct Krylov subspace methods for solving eigenvalue problems?

The answer to this question, in the case of Basic IFEAST, is yes. When any sort of Krylov subspace algorithm is used for solving the linear systems of equations for Basic IFEAST, then Basic IFEAST itself becomes a Krylov subspace method for solving eigenvalue problems [26]. Unlike other Krylov subspace methods for eigenvalue problems, though, it does not have to store the entire Krylov subspace basis in order to select the subspace that spans the eigenvectors of interest. Instead, it uses contour integration in order to form a basis for a subspace of the original Krylov subspace that spans only the eigenvectors whose eigenvalues are inside of the search contour.

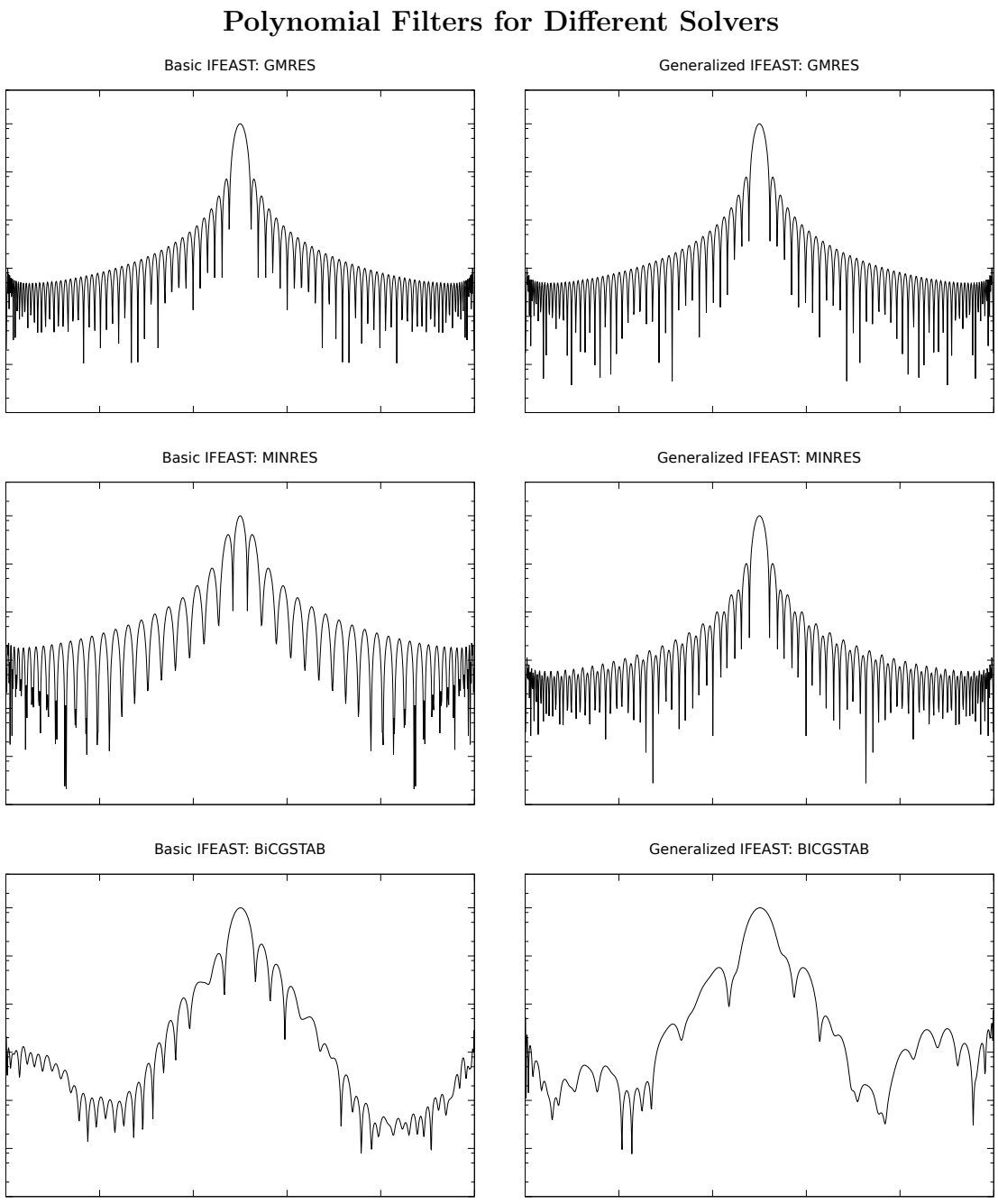


Figure 4.6. *A comparison of the polynomial filters that are produced by IFEAST when using several different linear system solving algorithms. All results use 100 iterations of their respective algorithms.*

In some cases it is possible to show that IFEAST is exactly equivalent to other well-known Krylov subspace methods for solving eigenvalue problems. Basic IFEAST implemented with block FOM is equivalent to block Arnoldi, Basic IFEAST implemented with block BiCG is equivalent to block Lanczos Biorthogonalization, and Basic IFEAST implemented with block GMRES is equivalent to block Arnoldi with Harmonic Ritz. The following subsections discuss these results in detail, and explain how they can inform a comparison between Basic IFEAST and other Krylov subspace eigenvalue algorithms. For the remainder of these sections I will refer only to "IFEAST", with the understanding that all of the following results regarding Krylov subspaces pertain only Basic IFEAST and not to Generalized IFEAST.

4.5.1 Contour Integration Selects Krylov Subspace

Standard Krylov eigenvalue solving methods (such as Lanczos and Arnoldi) work by building a basis V for the block Krylov subspace $\mathcal{K}_k(A, \tilde{X}^{(0)})$ of order k, using some (possibly random) initial guess $\tilde{X}^{(0)}$ for the eigenvectors. The column vectors of the matrix $V \in \mathbb{C}^{n \times m_0 k}$ span the subspace

$$\mathcal{K}_k(A, X^{(0)}) = span\{\tilde{X}^{(0)}, A\tilde{X}^{(0)}, A^2\tilde{X}^{(0)}, ..., A^{k-1}\tilde{X}^{(0)}\}. \tag{4.57}$$

I will consider a block Krylov subspace of block size m_0 (i.e. the dimension of the FEAST subspace) for the sake of easy comparison with the IFEAST algorithm.

Traditional Krylov methods then use the Rayleigh-Ritz procedure to form and solve a reduced-dimension eigenvalue problem in order to find approximate eigenpairs in the subspace $\mathcal{K}_k(A, X^{(0)})$

$$(V^H A V) X_V = (V^H V) X_V \Lambda. \tag{4.58}$$

For the sake of generality I do not assume that $V^H V = I$; the results that follow are true for any block Krylov algorithm, not just for algorithms such as Lanczos or Arnoldi that produce orthonormal V.

Let us assume that the order k of the Krylov subspace (4.57) is made as large as is practically possible. If the residuals of the approximate eigenpairs from the reduced problem (4.58) do not converge, then the method can be "restarted" by using a block of linear combinations of Ritz vectors $\tilde{X}^{(1)}$ from the solution of (4.58) as the starting vectors for building a new Krylov subspace $\mathcal{K}_k(A, X^{(1)})$ of order k. In Sections 4.5.2 and 4.5.4 I will discuss some ways that these restarting blocks may be chosen.

FEAST, when calculating the contour integration exactly, forms a subspace by applying a spectral projector to $\tilde{X}^{(0)}$; this subspace is then used to solve a reduced-dimension eigenvalue problem i.e.

$$Q = \rho(A)\tilde{X}^{(0)} = \frac{1}{2\pi i}\oint_{\mathcal{C}}(zI - A)^{-1}\tilde{X}^{(0)}dz, \tag{4.59}$$

$$(Q^H A Q)X_Q = (Q^H Q)X_Q\Lambda. \tag{4.60}$$

We can understand the relationship between IFEAST and traditional Krylov methods by considering what happens when the integrand $(zI - A)^{-1}\tilde{X}^{(0)}$ in (4.59) is evaluated approximately by using a Krylov subspace. We can rewrite the integral (4.59) as:

$$Q = \rho(A)\tilde{X}^{(0)} = \frac{1}{2\pi i}\oint_{\mathcal{C}} Y(z)dz, \tag{4.61}$$

where $Y(z)$ is the solution to the linear system

$$(zI - A)Y(z) = \tilde{X}^{(0)}. \tag{4.62}$$

If we use a Krylov subspace method to find an approximate solution to (4.62), then

$$Y(z) \approx \tilde{Y}(z) = V\tilde{Y}_V(z), \quad \tilde{Y}_V(z) \in \mathbb{C}^{m_0 k \times m_0}, \tag{4.63}$$

where V is the same Krylov subspace basis from equation (4.57), and $\tilde{Y}_V$ is an approximate solution to the least squares problem $(zI-A)VY_V(z) = \tilde{X}^{(0)}$. Importantly, the Krylov basis V is not a function of z, because the Krylov subspace that is generated by $(zI - A)$ depends only on the matrix A and not on the shift z. Because V is independent of z, we can rewrite the expression (4.61) for Q in such a way that the FEAST reduced-dimension eigenvalue problem (4.60) takes a familiar form. Rewriting the expression for Q, we get

$$Q = \frac{1}{2\pi i} \oint_{\mathcal{C}} \tilde{Y}(z)dz = VG_v, \tag{4.64}$$

with

$$G_v = \frac{1}{2\pi i} \oint_{\mathcal{C}} \tilde{Y}_V(z)dz \ \in \mathbb{C}^{m_0 k \times m_0}. \tag{4.65}$$

Then, the FEAST reduced eigenvalue problem (4.60) becomes

$$(G_v^H V^H AVG_v)X_Q = (G_v^H V^H VG_v)X_Q\Lambda. \tag{4.66}$$

Comparing (4.66) with (4.58) makes it clear that IFEAST itself is, in fact, a Krylov subspace method. The difference between IFEAST and more traditional Krylov methods is that IFEAST uses contour integration to select an ideally-suited linear combination of vectors from the Krylov basis V for finding the desired eigenvalues, without first having to solve a reduced eigenvalue problem in that basis.

Being able to select the desired eigenvalues in this way can have substantial benefits. One of the challenges in using Krylov subspaces is that finding certain eigenvalues, particularly interior eigenvalues or eigenvalues that are clustered closely together, can require a subspace basis V of very large dimension. Using a large-dimension sub-

space basis V entails large storage requirements for that basis, and a large computational cost for solving the corresponding reduced eigenvalue problem (4.58). When using IFEAST, on the other hand, the dimension of the reduced eigenvalue problem (4.66) is always m_0, which is substantially smaller than the dimension km_0 of the traditional reduced eigenvalue problem (4.58).

Moreover, when IFEAST is implemented with a linear system solver that uses a short recurrence relation (e.g. MINRES), then it can solve eigenvalue problems by using a Krylov subspace of arbitrarily large dimension without having to form and store a basis for that subspace; by using short recurrences, IFEAST can form the $n \times m_0$ matrix product $Q = VG_v$ without forming or storing either the $n \times km_0$ matrix V or the $km_0 \times m_0$ matrix G_v. Thus, eigenpairs that would previously have been difficult or impossible to obtain due to constraints on the dimension of V become much more tractable to calculate, and the spectrum slicing capability of FEAST is maintained by making it possible to selectively find specific eigenpairs anywhere in the spectrum.

The issues of large memory requirements and bad scaling[10] for the Rayleigh-Ritz procedure in traditional Krylov methods are often mitigated through the use of restarting strategies, which allow one to limit the dimension of the Krylov subspace that is used. These strategies carry with them the price of a reduced convergence rate, however; the smallest number of matrix-vector multiplications that is required for solving an eigenvalue problem with a traditional Krylov method is the number that is required when no restarts are used. For some eigenvalue problems the number of matrix multiplies that is needed when doing no restarts is still quite large; in cases like these, Basic IFEAST implemented with a short recurrence linear system solver

[10]The algorithmic complexity of solving the reduced eigenvalue problem is $O((km_0)^3)$ for a Krylov subspace of order k and block size m_0 for Arnoldi iterations, and many algorithms require costly basis reorthogonalization as well.

can use a large Krylov subspace without storing that subspace. This, combined with the use of natural parallelism in IFEAST, makes it possible to solve certain eigenvalue problems with a number of sequential matrix vector multiplications that is smaller than the number that would be required when using (for example) Arnoldi with no restarts. I provide an example of this in Section 4.6.6 (on page 141).

The relationship between IFEAST and traditional Krylov methods also offers a different perspective on achieving convergence when using restarts. In the context of IFEAST, a Krylov restart amounts to an approximate subspace iteration with $\rho(A)$ for a particular choice of contour $\mathcal{C}$. Using contour integration to choose the subspace with which to restart ensures that restarting will reliably result in convergence, with inequality (4.56) giving quantitative answers regarding whether or not restarting will result in convergence. IFEAST reverses the process that is used in other restarting strategies [70,88], in which the subspace that is used for restarting is determined *after* solving a reduced eigenvalue problem in the full Krylov subspace, rather than before.

I elaborate further on the relationship between IFEAST and traditional Krylov techniques in the following subsections, where I show how the implementation of IFEAST with particular linear system solvers is related to other Krylov subspace methods for solving eigenvalue problems. I show that, in the limit of exact integration, implementing IFEAST using the Full Orthogonalization Method (FOM) is equivalent to traditional explicitly restarted block Arnoldi, and that implementing IFEAST using GMRES is closely related to using Harmonic Rayleigh-Ritz for interior eigenvalue problems.

4.5.2 Restarted Block Arnoldi: IFEAST with FOM

The block Arnoldi method constructs an orthonormal basis $V \in \mathbb{C}^{n \times m_0 k}$ of block size m_0 and Krylov order k (for a total dimension of $m_0 k$), and then solves a reduced

eigenvalue problem from the Rayleigh-Ritz method in order to find estimates for the desired eigenvalues and eigenvectors, i.e.

$$HX_V = X_V\Lambda \tag{4.67}$$

$$H = V^H AV, \quad V^H V = I \tag{4.68}$$

where the column vectors of V span $\mathcal{K}_k(A, \tilde{X}^{(0)})$, $H \in \mathbb{C}^{m_0 k \times m_0 k}$ is block upper Hessenberg and $\tilde{X}^{(0)} \in \mathbb{C}^{n \times m_0}$ is the initial guess for the eigenvectors. If the residuals on the estimated eigenpairs (VX_V, Λ) are not good enough, then the method can be explicitly restarted by building a new Krylov subspace $\mathcal{K}_k(A, \tilde{X}^{(1)})$ using a new starting block $\tilde{X}^{(1)}$. The new starting block consists of linear combinations of the estimated eigenvectors, i.e.

$$\tilde{X}^{(1)} = VX_V M \tag{4.69}$$

where $M \in \mathbb{C}^{m_0 k \times m_0}$ gives the linear combinations that are used to determine each vector in the new starting block. A variety of different choices for M are possible [71]. A single iteration of IFEAST, when implemented with FOM, produces a new estimate for the eigenvectors of interest $\tilde{X}^{(1)}$ that is equivalent to expression (4.69) for a particular, natural choice of M.

Implementing IFEAST requires forming a subspace basis $Q \in \mathbb{C}^{n \times m_0}$ by evaluating the contour integral (4.61), which in turn requires solving (approximately, in this case) linear systems of the form (4.62). I restate these tasks (respectively) here, i.e.

$$Q = \rho(A)\tilde{X}^{(0)} = \frac{1}{2\pi i}\oint_{\mathcal{C}} \tilde{Y}(z)dz, \tag{4.70}$$

$$(zI - A)Y(z) = \tilde{X}^{(0)}, \tag{4.71}$$

$$\tilde{Y}(z) \approx Y(z). \tag{4.72}$$

FOM is used to approximate the solution to the linear system (4.71) by forming V using Arnoldi iterations, and then solving a projected linear system [73], i.e.

$$\tilde{Y}(z) = V\left(V^H(zI - A)V\right)^{-1} V^H \tilde{X}^{(0)}. \tag{4.73}$$

Because the linear system matrix $(zI - A)$ is just a shifted version of the original matrix A, the approximate solution $\tilde{Y}(z)$ can be written in terms of the block upper Hessenberg matrix that is generated by the block Arnoldi method, i.e.

$$\tilde{Y}(z) = V(zI - H)^{-1}V^H \tilde{X}^{(0)}. \tag{4.74}$$

Inserting this into the expression for the IFEAST subspace basis Q (4.70), it becomes clear that using FOM is equivalent to applying the FEAST filter function $\rho(\lambda)$ to the block upper Hessenberg matrix H from Arnoldi

$$Q = V\frac{1}{2\pi i}\oint_{\mathcal{C}}(zI - H)^{-1}dzV^H\tilde{X}^{(0)} = V\rho(H)V^H\tilde{X}^{(0)}. \tag{4.75}$$

This is equivalent to filtering out the components of the unwanted Arnoldi Ritz vectors from $\tilde{X}^{(0)}$, leaving only the Ritz vectors whose Ritz values are inside the contour $\mathcal{C}$ in the complex plane. We can see this by writing the eigenvalue decomposition of H and reordering its eigenvalues and eigenvectors so that the wanted eigenpairs (i.e. the ones whose eigenvalues are inside $\mathcal{C}$) are grouped together, i.e.

$$H = X_V \Lambda X_V^H, \tag{4.76}$$

$$X_V = \begin{bmatrix} X_w & X_u \end{bmatrix}, \quad \Lambda = \begin{bmatrix} \Lambda_w & 0 \\ 0 & \Lambda_u \end{bmatrix}, \tag{4.77}$$

and by writing the initial guess $X^{(0)}$ in terms of its Ritz vector components in the subspace spanned by V,

$$\tilde{X}^{(0)} = VX_wW + VX_uU, \tag{4.78}$$

where (X_w, Λ_w) are the m_0 wanted Ritz eigenpairs (i.e. the ones whose eigenvalues are inside $\mathcal{C}$ in the complex plane), (X_u, Λ_u) are the $(k-1)m_0$ unwanted Ritz eigenpairs, and W and U are the components of $\tilde{X}^{(0)}$ in terms of the wanted and unwanted Ritz eigenvectors (respectively). Rewriting (4.75) in these terms, we get

$$Q = V \begin{bmatrix} X_w & X_u \end{bmatrix} \begin{bmatrix} \rho(\Lambda_w) & 0 \\ 0 & \rho(\Lambda_u) \end{bmatrix} \begin{bmatrix} X_w & X_u \end{bmatrix}^H V^H (VX_wW + VX_uU) \tag{4.79}$$

$$= V(X_w\rho(\Lambda_w)W + X_u\rho(\Lambda_u)U). \tag{4.80}$$

IFEAST with FOM thus forms a subspace basis by filtering the Ritz values and vectors from the Arnoldi Rayleigh-Ritz matrix H; the components of $\tilde{X}^{(0)}$ in the direction of the wanted Ritz vectors are kept roughly the same, and the components of $\tilde{X}^{(0)}$ in the direction of the unwanted Ritz vectors are substantially reduced. When the contour integral in (4.75) is evaluated exactly, then $\rho(\Lambda_w) = I_{m_0 \times m_0}$ and $\rho(\Lambda_u) = 0_{(k-1)m_0 \times (k-1)m_0}$, and IFEAST forms and solves a reduced eigenvalue problem using only the Arnoldi Ritz vectors corresponding to the wanted Ritz values. The vectors that are used as the initial guess for the next IFEAST iteration, then, are just the normalized Arnoldi Ritz vectors corresponding to the Ritz values that are inside the contour $\mathcal{C}$ in the complex plane, i.e.

$$\tilde{X}^{(1)} = VX_w = VX_V \begin{bmatrix} I_{m_0 \times m_0} \\ 0_{(k-1)m_0 \times m_0} \end{bmatrix}. \tag{4.81}$$

IFEAST with FOM is equivalent, then, to performing block Arnoldi with a restart strategy that consists of selecting the desired Ritz vectors and discarding the rest.

In practice this restart strategy can be unreliable for obtaining eigenvalues in the interior of the spectrum. One perspective on why this happens is that the Rayleigh-Ritz procedure works well for resolving exterior eigenvalues, but not for resolving interior ones; restarting with Ritz vectors is thus unreliable for obtaining interior eigenvalues [56]. A remedy for this is to use the Harmonic Rayleigh-Ritz procedure [55, 56], wherein one solves a different reduced eigenvalue problem that more accurately obtains the eigenvalues that are located near some shift.

The fact that the restart strategy (4.81) is equivalent to using FOM with IFEAST suggests another perspective on why it is ineffective. Getting IFEAST to converge requires solving its associated linear systems such that their residuals are sufficiently small, and FOM does not minimize the linear system residual for a given subspace. Reliably achieving convergence for interior eigenpairs requires the use of a linear system solver that minimizes the linear system residual, such as GMRES or MINRES.

4.5.3 Restarted Lanczos Biorthogonalization: IFEAST with BiCG

A similar analysis to the one Section 4.5.2 can be performed for block Lanczos Biorthogonalization and two-sided[11] Basic IFEAST with block Biconjugate Gradients. Two-sided (generalized) IFEAST is summarized in Appendix D.

Block Lanczos (I will drop the "biorthogonalization" qualifier) generates *two* subspace bases $V, W \in \mathbb{C}^{n \times m_0 k}$ of block size m_0 and Krylov order k and then solves a reduced eigenvalue problem, i.e.

---

[11] Although this is often referred to as the "nonsymmetric FEAST algorithm", either one-sided or two sided FEAST can be applied to solving nonsymmetric eigenvalue problems. Two-sided FEAST is simply much more robust for nonsymmetric problems.

$$TX_V = X_V\Lambda \tag{4.82}$$

$$T^H X_W = X_W \Lambda^H \tag{4.83}$$

$$T = W^H AV, \quad W^H V = I, \tag{4.84}$$

where the columns of V span $\mathcal{K}(A, \tilde{X}_R^{(0)})$ and the columns of W span $\mathcal{K}(A, \tilde{X}_L^{(0)})$, with $\tilde{X}_R^{(0)}$ and $\tilde{X}_L^{(0)}$ being initial guesses for the right and left eigenvectors of interest, respectively. T, here, is a block tridiagonal matrix.

Lanczos, unlike Arnoldi, calculates estimates VX_V and WX_W for both the right and the left eigenvectors, and can be restarted just as Arnoldi can be if the corresponding eigenvector residuals are not sufficiently low by calculating new initial guesses

$$\tilde{X}_R^{(1)} = VX_V M \quad \text{and} \quad \tilde{X}_L^{(1)} = WX_W N, \tag{4.85}$$

with $M, N \in \mathcal{C}^{m_0 k \times m_0}$ determining the linear combinations of vectors that are used to form the columns of the initial guesses for restarting. A single iteration of two-sided IFEAST implemented with Biconjugate Gradients produces estimated right and left eigenvectors $\tilde{X}_R^{(1)}$ and $\tilde{X}_L^{(1)}$ that are equivalent to the restarting initial guesses in Equation (4.85).

Two-sided IFEAST refines *two* subspaces, one each for spanning the right and left eigenvectors of A:

$$Q_R = \rho(A)\tilde{X}_R^{(0)} = \frac{1}{2\pi i}\oint \tilde{Q}_R(z)dz \qquad Q_L = \rho(A)^H \tilde{X}_L^{(0)} = \frac{1}{2\pi i}\oint \tilde{Q}_L(z)dz \tag{4.86}$$

$$(zI - A)Q_R(z) = \tilde{X}_R^{(0)} \qquad (zI - A)^H Q_L(z) = \tilde{X}_L^{(0)} \tag{4.87}$$

$$\tilde{Q}_R(z) \approx Q_R(z) \qquad \tilde{Q}_L(z) \approx Q_L(z) \tag{4.88}$$

Biconjugate Gradients naturally solves two linear systems of equations at the same time (see Section 2.2.4), with the solution of one system of equations being approx-

imated in the subspace spanned by V and the other being approximated in the in the subspace spanned by W. The pairs of two-sided FEAST linear systems of equations (4.87) that share the same value of z can thus be approximated using block Biconjugate Gradients as

$$\tilde{Q}_R(z) = V(W^H(zI - A)V)^{-1}W^H\tilde{X}_R^{(0)} \tag{4.89}$$

$$= V(zI - T)^{-1}W^H\tilde{X}_R^{(0)} \tag{4.90}$$

$$\tilde{Q}_L(z) = W(V^H(zI - A)^HW)^{-1}V^H\tilde{X}_L^{(0)} \tag{4.91}$$

$$= W(z^*I - T^H)^{-1}V^H\tilde{X}_L^{(0)} \tag{4.92}$$

Substituting these into the integrals from Equation (4.86), we get

$$Q_R = \frac{1}{2\pi i}\oint V(zI - T)^{-1}W^H\tilde{X}_R^{(0)}dz \tag{4.93}$$

$$= V\rho(T)W^H\tilde{X}_R^{(0)} \tag{4.94}$$

$$Q_L = \frac{1}{2\pi i}\oint W(z^*I - T^H)^{-1}V^H\tilde{X}_L^{(0)}dz \tag{4.95}$$

$$= W\rho(T)^HV^H\tilde{X}_L^{(0)} \tag{4.96}$$

IFEAST with Biconjugate Gradients thus implicitly applies a spectral projector to the block tridiagonal Lanczos matrix T, which serves to select the Ritz values of T that are inside of the search contour. The selection of the corresponding Ritz vectors as initial guesses for restarting can be shown by applying the same steps that were used in Section (4.5.2) to each of the subspaces Q_R and Q_L.

4.5.4 Arnoldi with Harmonic Ritz: IFEAST with GMRES

Using GMRES with IFEAST is closely related to using the Harmonic Rayleigh-Ritz procedure with Arnoldi.

When using GMRES to approximately solve (4.71) for $Y(z)$, the approximate solution $\tilde{Y}(z)$ takes the form [73]

$$\tilde{Y}(z) = V\left(V^H(zI-A)^H(zI-A)V\right)^{-1}V^H(zI-A)^H\tilde{X}^{(0)}, \tag{4.97}$$

where V, again, is the block Arnoldi basis. The IFEAST subspace basis Q then becomes

$$Q = V\left(\frac{1}{2\pi i}\oint_{\mathcal{C}}\left[V^H(zI-A)^H(zI-A)V\right]^{-1}V^H(zI-A)^HVdz\right)\tilde{X}_V^{(0)}, \tag{4.98}$$

where $V\tilde{X}_V^{(0)} = \tilde{X}^{(0)}$ is the initial guess $\tilde{X}^{(0)}$ expressed in the Arnoldi basis V.

The integrand in (4.98) is equivalent to the matrix that one arrives at when using Harmonic Rayleigh-Ritz with Arnoldi. With Harmonic Rayleigh-Ritz, one seeks to find approximations for the eigenvalues that are near some shift $z \in \mathbb{C}$, using the subspace basis V. This is done by solving the reduced, generalized eigenvalue problem [55, 56]

$$A_V(z)X_V(z) = B_V(z)X_V(z)(zI - \Lambda(z)), \tag{4.99}$$

$$A_V(z) = V^H(zI-A)^H(zI-A)V, \quad B_V(z) = V^H(zI-A)^HV, \tag{4.100}$$

where $VX_V(z)$ are now the Harmonic Ritz vectors, and $\Lambda(z)$ are the Harmonic Ritz values. In most applications the shift z is taken to be a fixed parameter, but here we

are considering a case where it will vary, making the projected matrices $A_V(z)$ and $B_V(z)$, and the Harmonic Ritz vectors and values $X_V(z)$ and $\Lambda(z)$, into matrix-valued functions of the shift. Like any generalized eigenvalue problem, (4.99) can be written as a standard, non-symmetric eigenvalue problem with a corresponding eigenvalue decomposition, i.e.

$$B_V^{-1}(z)A_V(z) = X_V(z)(zI - \Lambda(z))X_V^{-1}(z). \tag{4.101}$$

If we note that

$$\left[B_V^{-1}(z)A_V(z)\right]^{-1} = \left[V^H(zI-A)^H(zI-A)V\right]^{-1} V^H(zI-A)^H V, \tag{4.102}$$

then we can use this combined with Equation (4.101) in order to write the expression for Q (4.98) in terms of the Harmonic Rayleigh-Ritz eigenvalue decomposition:

$$Q = V\left(\frac{1}{2\pi i}\oint_{\mathcal{C}}\left[B_V^{-1}(z)A_V(z)\right]^{-1}dz\right)\tilde{X}_V^{(0)}, \tag{4.103}$$

$$= V\left(\frac{1}{2\pi i}\oint_{\mathcal{C}}\left[zI - X_V(z)\Lambda(z)X_V^{-1}(z)\right]^{-1}dz\right)\tilde{X}_V^{(0)}. \tag{4.104}$$

Generating the IFEAST subspace basis by using GMRES is thus equivalent to using contour integration to filter the initial guess by using Arnoldi Harmonic Ritz values and vectors. Unlike with FOM, however, the resulting contour integral is not equivalent to applying the usual FEAST spectral filter $\rho(\lambda)$ to a projected matrix. Instead, the integration in (4.104) is the contour integral of the resolvent of a *nonlinear* eigenvalue problem, where the eigenvalues and eigenvectors are functions of the complex variable z that are derived from the Harmonic Rayleigh-Ritz procedure.

4.6 Examples

This section will describe several example problems in order to illustrate the properties of the IFEAST algorithm. Of particular interest are situations in which the behavior of IFEAST deviates from that of the standard FEAST algorithm, usually due to limitations that are inherent to using polynomial filters for solving eigenvalue problems. The problems that we will consider are:

Standard uniform problem: a standard, Hermitian eigenvalue problem with 1000 eigenvalues uniformly distributed over the interval $[1000, 1001]$. The uniform distribution of the eigenvalues allows us to probe the properties of IFEAST that are independent of any effects that may arise due to the relative locations of the eigenvalues in a matrix's spectrum.

Generalized uniform problem: a generalized, Hermitian eigenvalue problem with the same eigenvalues as the *standard uniform problem*. This problem allows us to compare the Basic IFEAST algorithm and the Generalized IFEAST algorithm, which is the more efficient algorithm to use for solving generalized eigenvalue problems.

Standard nonuniform problems: a collection of standard, Hermitian eigenvalue problems with 1000 eigenvalues in the interval $[10^{-5}, 1]$. All of the matrices in this collection have the same largest-magnitude and smallest-magnitude eigenvalues (i.e. 1 and 10^{-5}), and therefore have the same condition number. Their eigenvalues are distributed nonuniformly over the interval $[10^{-5}, 1]$ such that there is a higher density of eigenvalues near 10^{-5} than there is near 1. For some of these matrices the eigenvalue density varies much more than it does for others. These examples allow us to examine the effects that the distribution of a matrix's spectrum has on the convergence of IFEAST.

Grcar problem: a nonsymmetric Toeplitz matrix of dimension 100. The eigenvalues of the Grcar matrix are known to be particularly sensitive to perturbations [98], which allow us to demonstrate how IFEAST behaves when applied to problems that are both nonsymmetric and particularly challenging.

Ga41As41H72: a standard Hermitian eigenvalue problem of dimension 268,096 generated by the electronic structure code PARSEC [45], and provided by the University of Florida Sparse Matrix Collection [15]. The relatively large dimension of this problem allows us to draw a stark contrast between the performance of standard FEAST and IFEAST.

The first four of these problems are simple, artificial matrix pencils that are designed to highlight specific qualities of the algorithms under consideration. The last problem is a natural example problem that is derived from practical application.

Unless otherwise noted, all of the results in this section for the IFEAST algorithm are produced by using the Generalized variation of the IFEAST algorithm. The reported eigenvector residuals for IFEAST are the largest residuals for the estimated eigenpairs whose eigenvalues are inside of the search contour.

4.6.1 Convergence: n_c and m_0

We can examine the behavior of IFEAST with respect to the dimension of the FEAST subspace, m_0, and the number of integration quadrature points, n_c, by calculating the lowest 10 eigenvalues of the *standard uniform problem*.

Figure 4.7 shows the eigenvector residual versus IFEAST iteration number for several values of n_c, with m_0 held fixed. Results are shown using both standard FEAST and IFEAST. For standard FEAST, increasing the value of n_c always produces an appreciable improvement in the rate of convergence. The same is not true for IFEAST; increasing n_c from 2 to 6 improves the rate convergence substantially, but increasing n_c from 6 to 12 does not produce any change in the rate of convergence.

Figure 4.8 shows the eigenvector residual versus IFEAST iteration number for several values of m_0, with n_c held fixed. Results are again shown using both standard FEAST and IFEAST. The rate of convergence for standard FEAST improves a lot when changing m_0 from 15 to 30, but it appears to improve much less when changing m_0 from 30 to 60. This is mostly just because standard FEAST already converges very quickly for $m_0 = 30$; the change from $m_0 = 30$ to $m_0 = 60$ reduces the required number of iterations for convergence by 33%. The rate of convergence for IFEAST also appears to change little when m_0 goes from 30 to 60, but in this case the reason is not because IFEAST is already converging quickly.

These results can be explained by examining the respective polynomial or rational filters of the IFEAST or standard FEAST algorithms. Figure 4.9 shows the polynomial and rational filter values plotted in sorted order (from largest-magnitude to smallest-magnitude) for IFEAST and standard FEAST, for the same values of n_c that were used to generate the results in Figures 4.7 and 4.8. Plot markers indicate the locations of the $(m_0 + 1)^{\text{th}}$ largest value of each filter, the value of which indicates the amount by which we would expect the eigenvector error to be reduced at each FEAST or IFEAST iteration. Examining the filter functions shows clearly why changing n_c from 2 to 6 improves the convergence rate of IFEAST, but changing n_c from 6 to 12 does not. The value of the IFEAST polynomial filter generally drops more quickly for $n_c = 6$ than it does for $n_c = 2$, but the value of the polynomial filter for $n_c = 6$ is exactly the same as for $n_c = 12$, thus producing no improvement in convergence. This stands in stark contrast to standard FEAST, for which the value of the rational filter function always drops more quickly with increasing n_c.

Figure 4.9 appears to indicate that, for $m_0 = 15$, increasing the value of n_c from 2 to 6 (or 12) should not change the rate of convergence. Figure 4.10 shows the same experiment as Figure 4.9, this time repeated with $m_0 = 15$. With $m_0 = 15$ the rate of convergence actually becomes worse when increasing n_c from 2 to 6, and it is roughly

the same for $n_c = 2$ and $n_c = 12$. The $n_c = 12$ case *appears* to converge more quickly, but that is just because it enters the regime of linear convergence more quickly; the slope of the convergence trajectory in Figure 4.9 is roughly the same for $n_c = 2$ and $n_c = 12$. This is a convenient illustration of the fact that, as discussed in Section 4.4.3, examining the values of the IFEAST polynomial function for a uniform initial guess vector can only offer qualitative insights into the behavior of IFEAST.

Convergence for Different Values of n_c

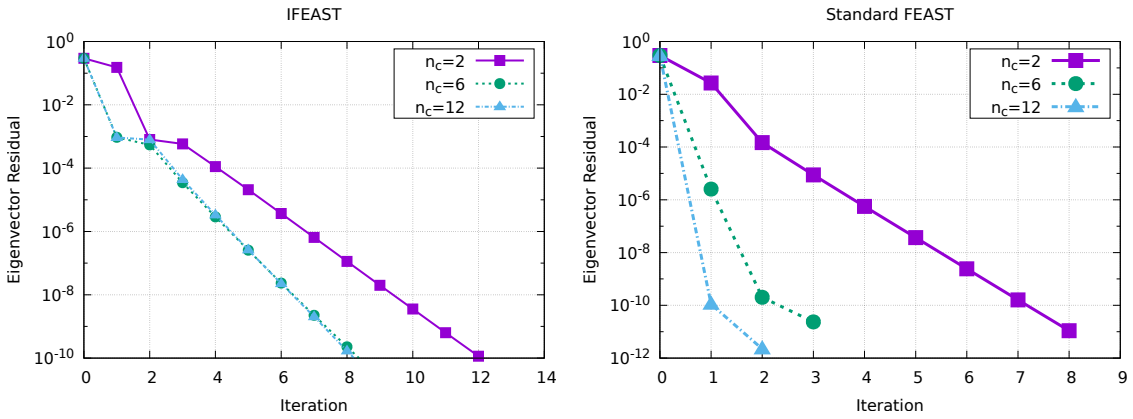


Figure 4.7. *Eigenvector residual versus FEAST iteration for calculating the 10 lowest eigenvalues of the standard uniform problem, for several values of n_c. For IFEAST $m_0 = 30$ and for standard FEAST $m_0 = 15$. These values of m_0 were chosen to provide clear illustrations of the effect of changing n_c. IFEAST used 50 iterations of MINRES at each outer IFEAST iteration for solving each of the n_c linear systems of equations.*

4.6.2 Convergence: middle of the spectrum

The behavior of the IFEAST algorithm changes depending on where the search contour is located relative to the rest of the spectrum of a matrix. Eigenvalues that are in the middle of the spectrum are generally more difficult to calculate than are eigenvalues that are at the edges of the spectrum. This is also true of the standard FEAST algorithm, but the severity of the change in performance is much greater for

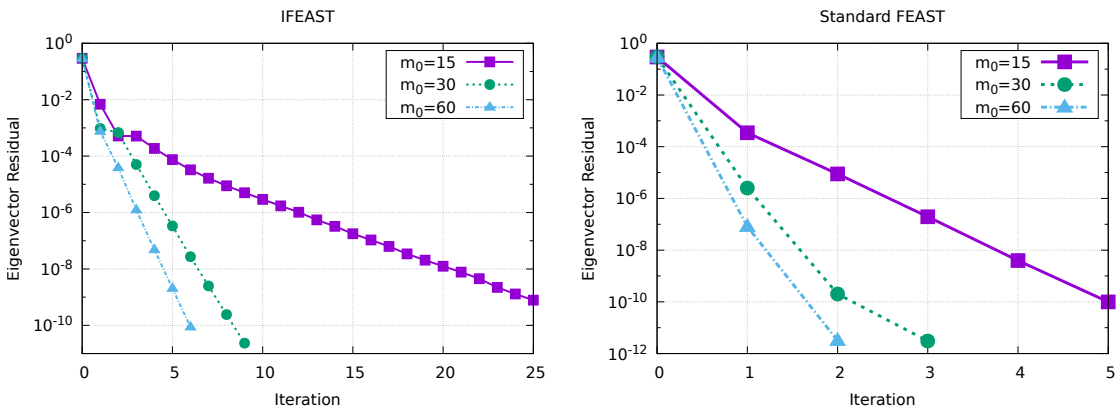


Figure 4.8. *Eigenvector residual versus FEAST iteration for calculating the lowest 10 eigenvalues of the standard uniform problem, for several values of m_0 and with $n_c = 6$. IFEAST used 50 iterations of MINRES at each outer IFEAST iteration for solving each of the n_c linear systems of equations.*

IFEAST than it is for standard FEAST. The IFEAST polynomial filter is much less responsive to changes in the value of n_c for search contours that are the middle of a spectrum, owing largely to the fact that it is more difficult to refine a polynomial approximation for the indicator function when all of the samples points that are to be set to 1 are in the middle of the collection of sample points, rather than at the edge. This is illustrated in Figure 4.11, which repeats the experiment from Figure 4.7 with the search contour in the middle of the spectrum rather than at the edge.

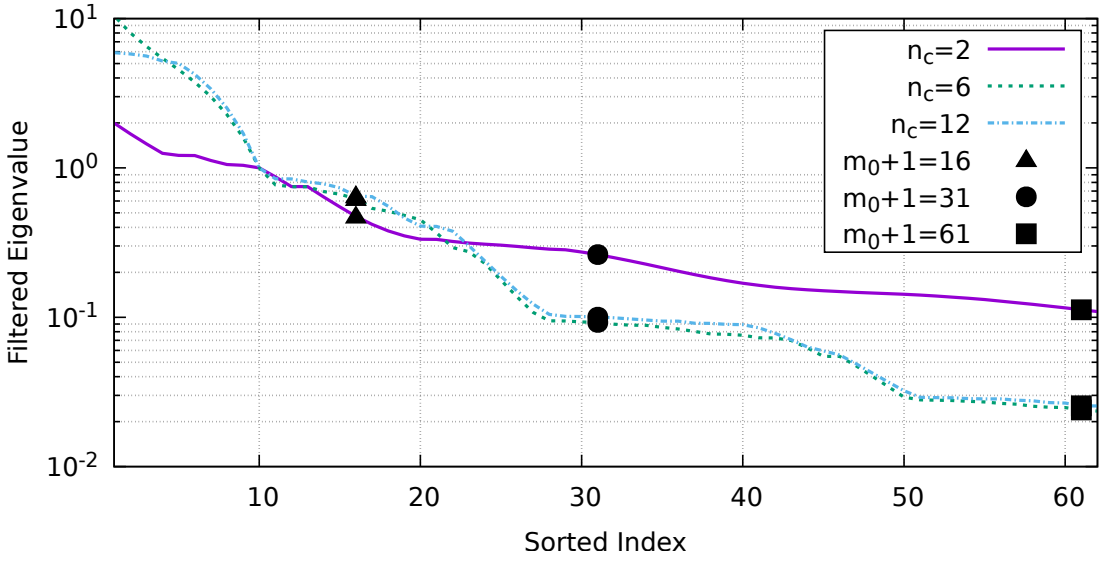
The result is that convergence becomes worse for increasing m_0, rather than better. This is due to the fact that the linear system shifts z_k are brought closer to the eigenvalues of the matrix as n_c is increased, but the number of linear system iterations is kept the same, and is not quite large enough for IFEAST to converge easily.

4.6.3 Eigenvalue Density and Linear System Conditioning

Given that the IFEAST algorithm solves eigenvalue problems by iteratively solving linear systems of equations, it is natural to assume that the condition number of the

Sorted Polynomial and Rational Filter Values: Various n_c

Polynomial Filter / IFEAST



Rational Filter / Standard FEAST

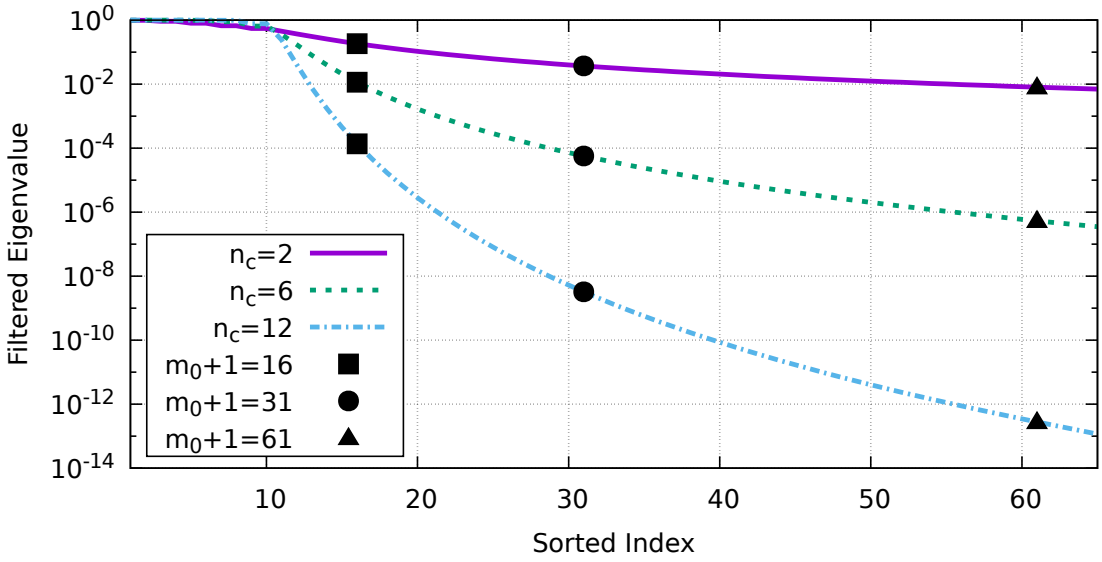


Figure 4.9. *Polynomial and rational filter values plotted in sorted order from largest magnitude to smallest magnitude, evaluated only at the eigenvalues of the standard uniform problem. Plotted filters are for selecting the lowest 10 eigenvalues. The polynomial filters are degree 50. Both plots show filter values for n_c =2, 6, and 12; the curves for $n_c = 6$ and $n_c = 12$ are identical for the polynomial plot, and therefore are overlaid on top of one another. Plot markers indicate the $(m_0+1)^{th}$ largest value of filters, for the same values of m_0 that are used in Figure 4.8.*

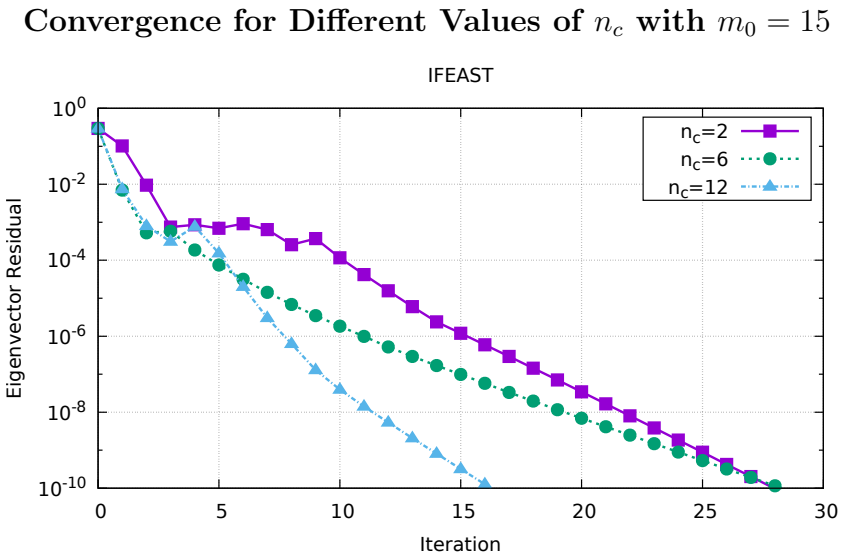


Figure 4.10. *Eigenvector residual versus IFEAST iteration for calculating the 10 lowest eigenvalues of the standard uniform problem, for several values of n_c and with $m_0 = 15$. IFEAST used 50 iterations of MINRES at each outer FEAST iteration for solving each of the n_c linear systems of equations.*

matrices involved should have an impact on the performance of IFEAST. This is true, but the relationship between the condition number of the matrices involved and the performance of IFEAST is somewhat indirect. For a given matrix A it is not the condition number of A that is important; rather, it is the condition numbers of the shifted linear systems $(z_k I - A)$ that matter. This has an effect that may at first seem somewhat counter-intuitive: using IFEAST to solve an eigenvalue problem for a matrix A that is very poorly-conditioned can be easy, provided that the eigenvalues of A are close to being uniformly-distributed. On the other hand, using IFEAST to solve an eigenvalue problem for a matrix A that is well-conditioned can be very difficult if the eigenvalues of A are distributed very unevenly.

We can illustrate this effect by considering the *standard nonuniform problems:* three dimension 1000 Hermitian matrices (which I will refer to here as Matrix 1, Matrix 2, and Matrix 3) whose eigenvalues all lie in the interval $[10^{-5}, 1]$. Importantly, all three matrices have the same largest-magnitude and smallest-magnitude eigenval-

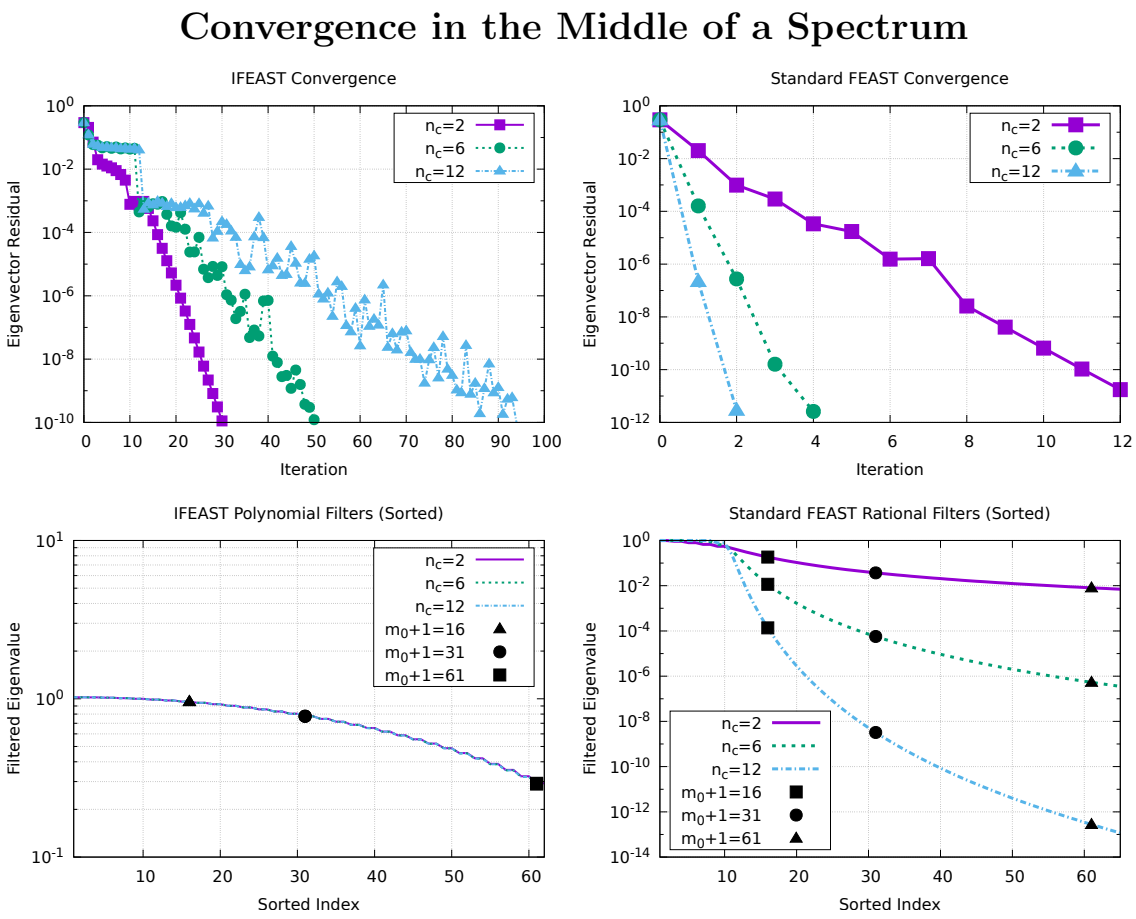


Figure 4.11. *Plots showing the convergence of FEAST and IFEAST for calculating 10 eigenvalues in the middle of the spectrum of the standard uniform problem. IFEAST used 50 MINRES iterations at each outer IFEAST iteration. Top plots show the convergence of the eigenvector residual versus iteration number, and bottom plots show the corresponding rational or polynomial filter functions. The bottom left plot shows three filters, but they can not be distinguished from one another because they have approximately the same values.*

ues: 1 and 10^{-5}, respectively. As a result, all three matrices have the same condition number.

These matrices are all generated in such a way that their eigenvalues are distributed nonuniformly; their eigenvalues are clustered more closely to 10^{-5} than they are to 1. The rate of increase in the eigenvalue density near 10^{-5} is greater for Matrix 2 than it is for Matrix 1, and it is greater for Matrix 3 than it is for Matrix 2. The densities of the eigenvalues on the real number line for each of these matrices are plotted in Figure 4.12.

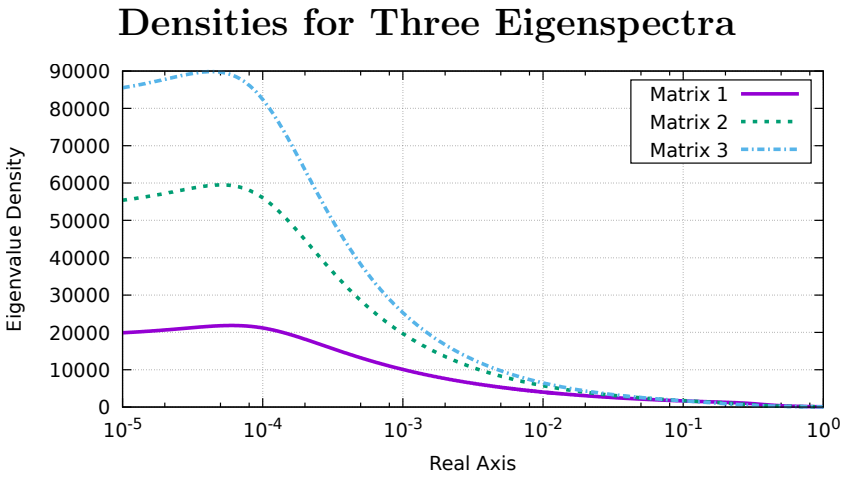


Figure 4.12. *Density of eigenvalues on the real number line for three different matrix eigenvalue spectra. Each spectrum has 1000 eigenvalues distributed according to the densities shown. All spectra have their smallest-magnitude eigenvalue at* 10^{-5} *and their largest-magnitude eigenvalue at* 1.0.

When using IFEAST to calculate the 10 smallest-magnitude eigenvalues for each of these matrices, the solution for Matrix 1 (which has the least density variation in its spectrum) converges the most quickly, and the solution for Matrix 3 (which has the most density variation in its spectrum) converges the most slowly. The rate of convergence for Matrix 2 falls in between. The same is not true when we use standard FEAST; with standard FEAST the calculations for all three matrices converge at similar rates, and the matrices with greater density variation actually

converge slightly faster than the matrices with lesser density variation. These results are illustrated in Figure 4.13.

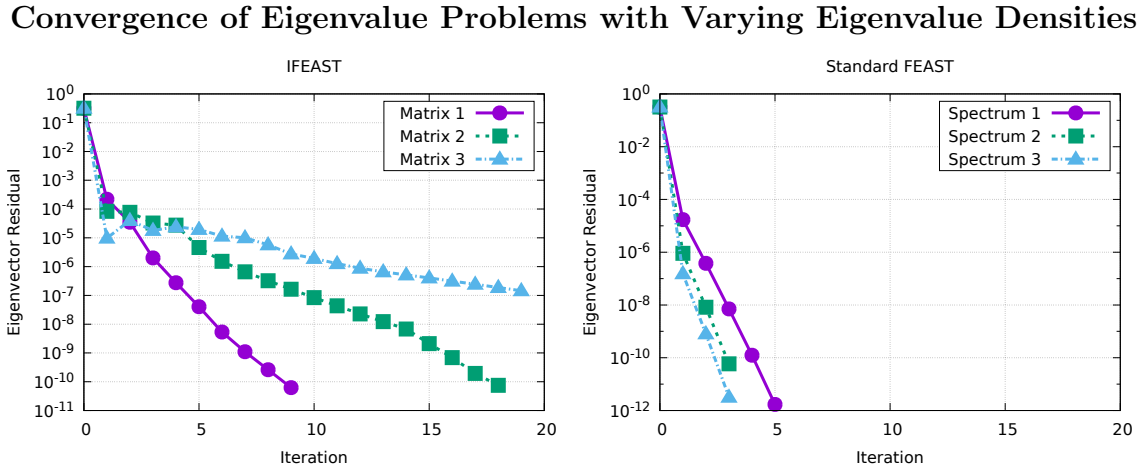


Figure 4.13. *Eigenvector residual versus IFEAST iteration for calculating the 10 smallest-magnitude eigenvalues of several* 1000×1000 *Hermitian matrices with varying eigenvalue densities (densities shown in Figure 4.12). Each of these results is produced by using IFEAST with* $n_c = 4$*,* $m_0 = 15$*, and 100 MINRES iterations at each IFEAST iteration.*

The results in Figure 4.13 can be explained in terms of conditioning of the shifted linear systems $(z_k I - A)$. For Matrix 1, which has the smallest amount of eigenvalue density variation, the shifts z_k are relatively far away from the eigenvalues of A, and so the shifted linear systems $(z_k I - A)$ are well-conditioned. For Matrix 3, which has the largest amount of density variation, the shifts z_k are much closer to the eigenvalues of A, and the condition number of $(z_k I - A)$ is correspondingly much worse. Table 4.1 summarizes the original and shifted condition numbers for each of the three matrices.

The eigenvalue search contours and quadrature shifts z_k are illustrated in Figure 4.15 (on page 136). Each contour in that figure encloses the same number of eigenvalues at the lower edge of the spectrum. The more the density of the corresponding matrix spectrum increases near 10^{-5}, the smaller the corresponding search contour

Matrix Condition Numbers

	$\kappa(A)$	$\max_k \kappa(z_k I - A)$
Matrix 1	10^5	7.8×10^3
Matrix 2	10^5	7.9×10^4
Matrix 3	10^5	3.2×10^5

Table 4.1. *Comparison of the condition numbers for Matrices 1, 2, and 3. The first column shows the condition number of the original matrix, and the second column shows the worst condition number for all of the IFEAST shifted matrices* $(z_k I - A)$*. The condition numbers of the shifted matrices vary considerably because the size of the corresponding IFEAST search contours changes due to differences in density of the eigenvalue spectrum.*

needs to be in order to select only the 10 smallest-magnitude eigenvalues, and the closer the shifts z_k get to the real axis (where all of the eigenvalues are located).

The variation in the condition number of the shifted matrices has a strong effect on the convergence rate of IFEAST, which solves linear systems of equations iteratively. This is shown in Figure 4.14 (on page 135), which plots the average linear system residual at each IFEAST iteration. The matrices with larger variations in their eigenvalue density, and therefore worse condition numbers for the shifted matrices, have their linear systems of equations solved less accurately when using using the same, constant number of MINRES iterations. This effect does not appear for standard FEAST, which solves the linear systems of equations directly by using factorization-based methods.

The effect of varying density (and therefore contour size) can also be explained in terms of the polynomial and rational filters that are implicitly generated by IFEAST and standard FEAST. The polynomial filter, in particular, becomes substantially worse as the variation in density increases, in the sense that its value drops less quickly outside of the search region. The rational filter actually becomes better as the variation in density increases, although the rational filters for all three matrices in this case are very similar. This is illustrated in Figure 4.16 (on page 137), which

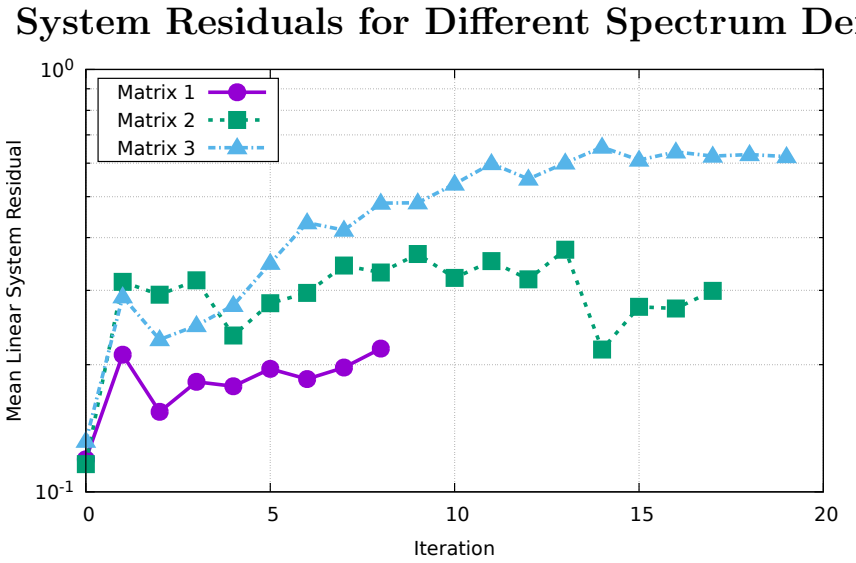


Figure 4.14. *Mean linear system residual (i.e. averaged over all shifted linear systems and all right hand sides) versus IFEAST iteration. Residuals are measured relative to the norm of the right hand side.*

shows the polynomial and rational filters evaluated at the eigenvalues of each of the three matrices, plotted in sorted order from largest to smallest.

4.6.4 Generalized IFEAST vs. Basic IFEAST

The purpose of the Generalized IFEAST algorithm is to be able to solve generalized eigenvalue problems using by using IFEAST with a constant number of linear system iterations at each outer IFEAST iteration. We can demonstrate the effectiveness of this approach by solving the *generalized uniform problem* for the middle 10 eigenvalues in the spectrum, using both Basic IFEAST and Generalized IFEAST with 100 MINRES iterations at each outer IFEAST iteration. Figure 4.17 (on page 137) shows the resulting plot of the eigenvector residual versus iteration number. As anticipated, the Basic IFEAST algorithm fails to converge, whereas the Generalized IFEAST algorithm converges linearly to the solution.

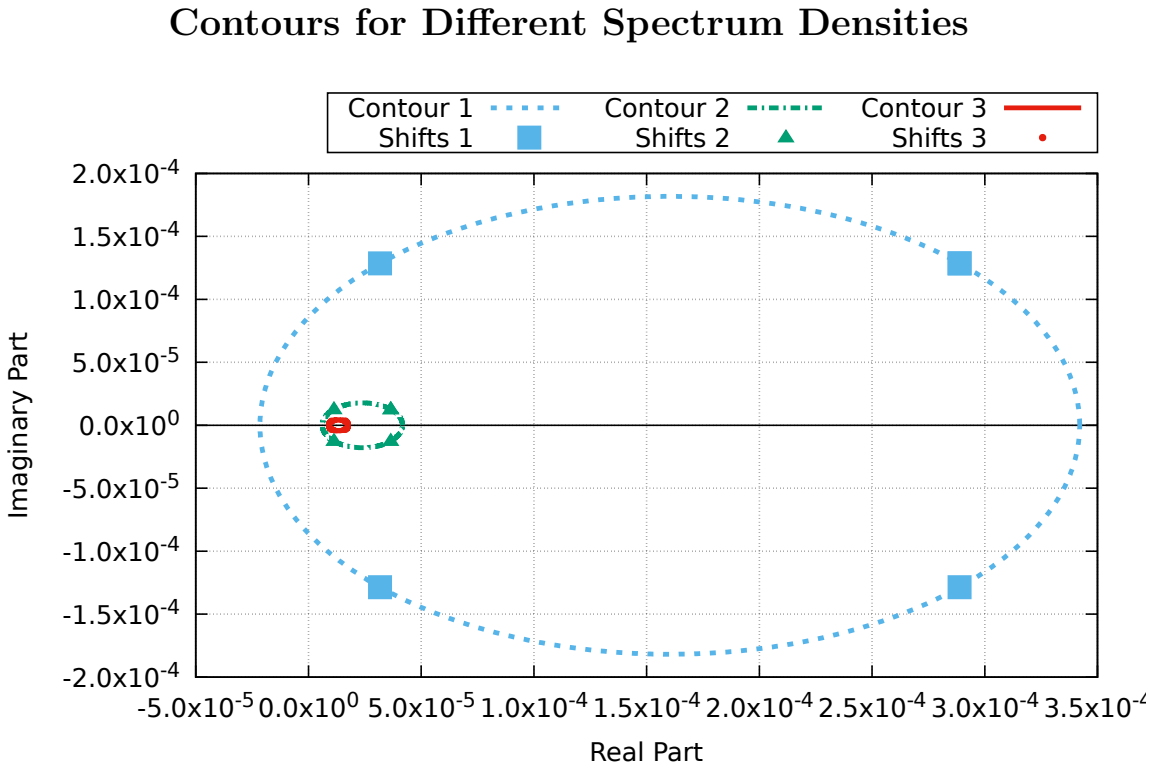


Figure 4.15. *Integration contours for the results shown in Figure 4.13; Contour 1 corresponds to Matrix 1, Contour 2 corresponds to Matrix 2, and Contour 3 corresponds to Matrix 3. Each contour is perfectly circular (they appear ellipsoid due to the plot bounds) and encloses the 10 smallest-magnitude eigenvalues of its respective spectrum. The contours differ in size due to the varying densities of the spectra that they are enclosing.*

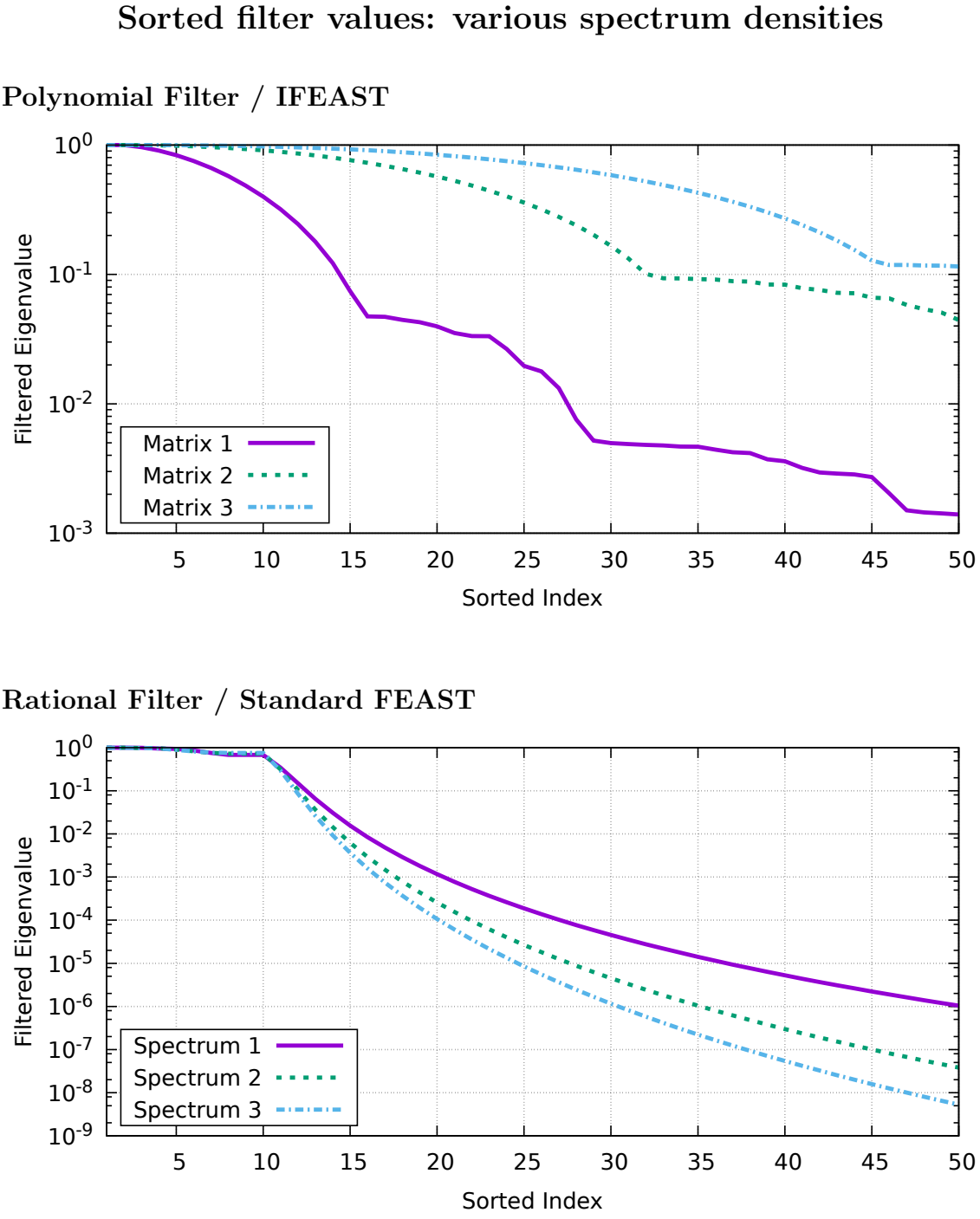


Figure 4.16. *Polynomial and rational filter values plotted in sorted order from largest magnitude to smallest magnitude, evaluated only at the eigenvalues of several* 1000×1000 *Hermitian matrices with varying eigenvalue densities (shown in Figure 4.12). The polynomial filters are degree 100.*

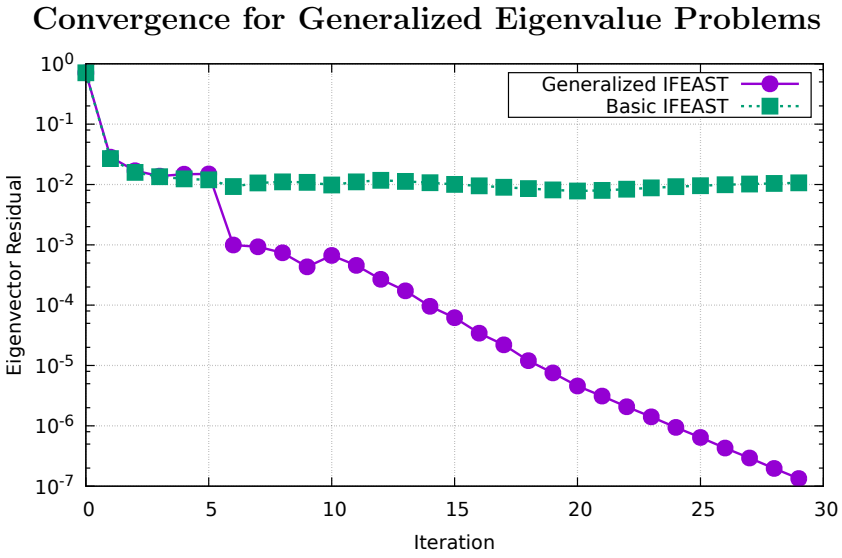


Figure 4.17. *Eigenvector residual versus iteration number for calculating the middle 10 eigenvalues of the generalized uniform problem with Simple IFEAST and Generalized IFEAST. Both algorithms use 100 MINRES iterations for solving each linear system at each outer IFEAST iteration. These results were obtained using* $m_0 = 15$ *and* $n_c = 4$.

4.6.5 A nonsymmetric problem

We can use the *Grcar* problem as a demonstration of Generalized IFEAST for nonsymmetric problems, and particularly for problems that are difficult to diagonalize.

First, let us consider using the two-sided Generalized IFEAST with BiCGSTAB to calculate eigenvalues that are at the edge of the spectrum. Two-sided Generalized IFEAST is described in Algorithm 11 on page 190 in the appendix. Figure 4.18 (on page 140) shows the results of these experiments in 3 plots; each plot shows the locations of the eigenvalues calculated with ZGGEV from LAPACK and the locations of the estimated eigenvalues from IFEAST, for different values of n_c and different numbers of BiCGSTAB iterations at each IFEAST iteration.

Setting $n_c = 2$ and using 30 BiCGSTAB iterations successfully identifies all of the eigenvalues inside of the contour, and several of the eigenvalues outside of the contour.

Setting $n_c = 4$ and using 30 BiCGSTAB iterations fails to accurately identify most of the eigenvalues inside the contour, and it identifies none of the eigenvalues outside of the contour. The difference between these two cases is the locations of the quadrature points: for $n_c = 2$ the quadrature points are relatively far from the eigenvalues of the matrix, whereas for $n_c = 4$ one of the contour points is almost in the same location as one of the eigenvalues. This is reflected in the largest condition number of the IFEAST linear systems of equations, which is 2.8×10^9 for $n_c = 2$ and 1.2×10^{15} for $n_c = 4$. Successful convergence to the eigenvalues of interest can be recovered by increasing the number of BiCGSTAB iterations to 50 in order to solve the IFEAST linear systems more accurately.

Notably for this problem, the eigenvector residuals from the solutions given by ZGGEV (which uses the QZ algorithm) are not particularly good[12]; the lowest one is 1.2×10^{-5}, whereas IFEAST is able to produce solutions that have a maximum eigenvector residual of 5.8×10^{-6} (for eigenvalues inside the contour). The inaccuracy of the ZGGEV solutions is visible in the lower-middle portion of the Grcar spectrum, which should mirror the top-middle portion, but is instead somewhat jumbled. These are challenging eigenvalues to calculate, and Generalized IFEAST struggles with them as well. Figure 4.19 (on page 141) shows the results of using Generalized IFEAST to calculate some of the eigenvalues in this region. IFEAST fails to converge even with 70 iterations of BiCGSTAB. In order for IFEAST to converge it is necessary to use the more robust GMRES instead. The challenge here is partly the condition number of the linear systems, which is 3.9×10^{12} at its largest, and partly the fact that we are calculating interior eigenvalues, which are inherently more challenging. Although IFEAST struggles with this problem, when it does converge its eigenvalue solutions

[12]ZGEEV, which uses QR iterations instead, would provide lower residuals; I provide comparison with ZGGEV, however, because this is the routine that I use for solving Rayleigh-Ritz problems inside of the IFEAST implementation that was used for these results.

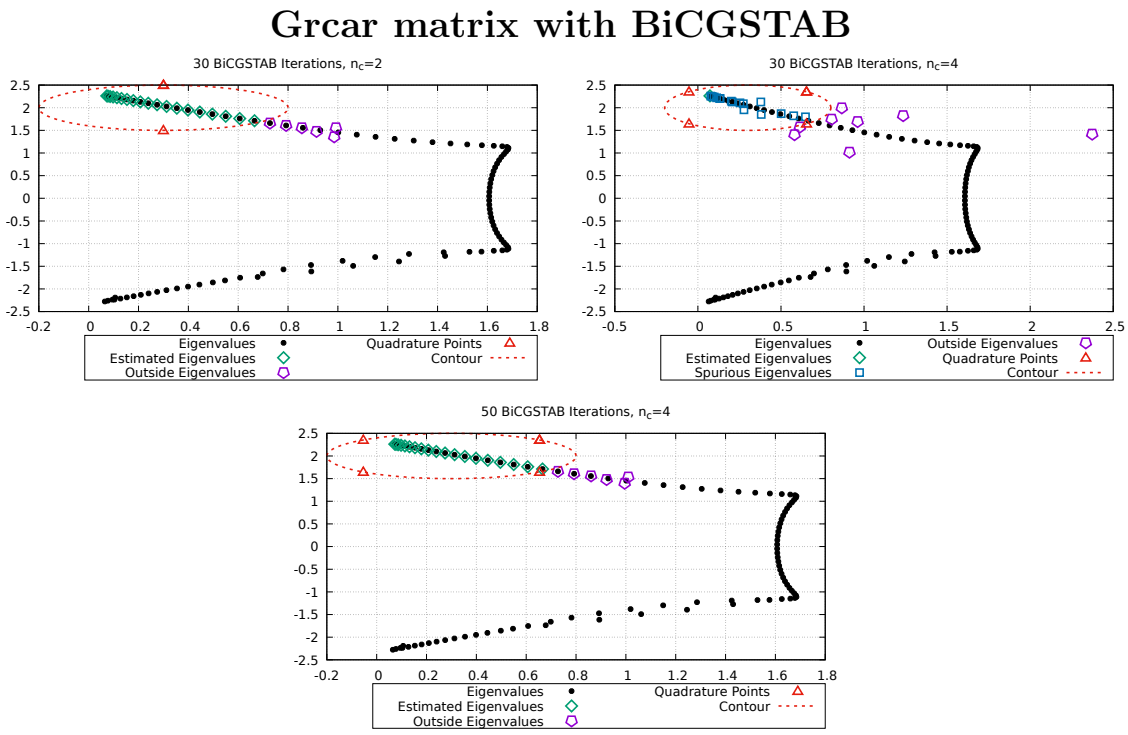


Figure 4.18. *Plots showing the locations in the complex plane of the estimated eigenvalues for the Grcar matrix, along with the IFEAST search contour and quadrature points, for different values of n_c and numbers of BiCGSTAB iterations. Black dots labeled "Eigenvalues" indicate the eigenvalue estimates returned by ZGGEV. Subspace dimension m_0 is 25. Plot markers labeled "Eigenvalues" are the locations of the eigenvalues as calculated by ZGGEV with LAPACK. "Estimated Eigenvalues" are the estimated eigenvalues from Generalized IFEAST that are inside the search contour, and eigenvalues labeled as "spurious" failed to converge below a certain tolerance, in this case 10^{-2}. The eigenvector residual tolerance for all three calculations was set at 10^{-5}; top left required 37 iterations to converge, bottom middle required 17 iterations to converge, and top right did not converge in 60 iterations.*

are visibly more correct than the corresponding ZGGEV solutions. Even so, these results suggest that larger problems of equal difficulty may be prohibitively difficult to solve without preconditioners, because using large numbers of GMRES iterations for large eigenvalue problems will usually be too time-consuming and memory-intensive.

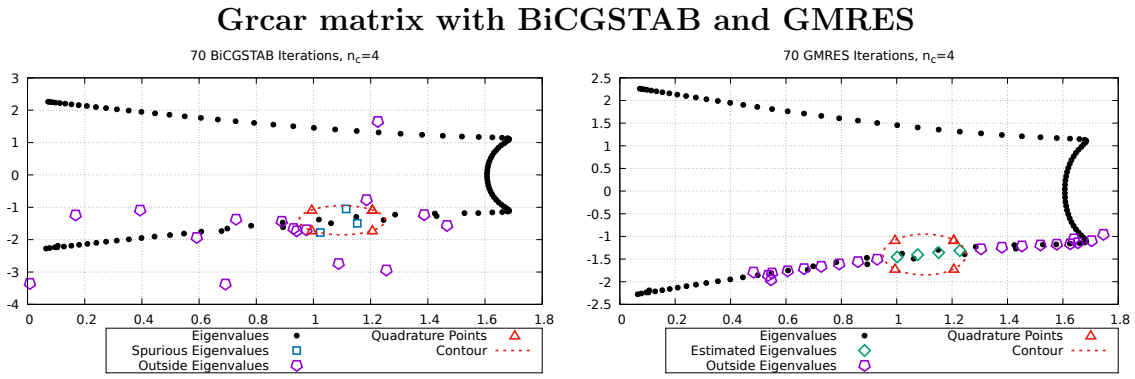


Figure 4.19. *Plots of the estimated eigenvalues for the Grcar matrix from Generalized IFEAST and ZGGEV again, this time for a different search contour location. Black dots labeled "Eigenvalues" indicate the eigenvalue estimates returned by ZGGEV, other eigenvalues are the estimates returned by IFEAST. Results are shown using both BiCGSTAB and GMRES as linear system solvers, with* $m_0 = 25$ *in both cases. IFEAST with GMRES required 21 iterations to converge below* 10^{-5} *for the eigenvector residual, and IFEAST with BiCGSTAB did not converge in 60 iterations.*

4.6.6 A Practical Example

To give a sense of how IFEAST behaves in a real-world context, we can apply it to the Ga41As41H72 problem. This is an eigenvalue problem that is derived from electronic structure calculations, which use the eigenvalue and eigenvector solutions to predict the optical, chemical, and electrical properties of molecules and atoms. The usual task in a problem like this is to calculate some number of the lowest eigenvalues and their corresponding eigenvectors, as these are the quantities that predict the ground state properties of a physical system. In this case I calculate the 50 lowest eigenvalues and corresponding eigenvectors of Ga41As41H72 such that the eigenvector residual norms are below 10^{-10}.

Table 4.2 compares the number of iterations and amount of time required by the FEAST algorithm for calculating the lowest 50 eigenvalues of Ga41As41H72, using both IFEAST with BiCGSTAB and standard FEAST with the PARDISO direct linear system solver [78]. Although IFEAST requires 5 more iterations than standard FEAST does, it takes substantially less time to converge. The essential issue is that the factorization of the shifted FEAST matrices requires a lot of time and a lot of memory. It requires so much memory, in fact, that it was not possible to save and reuse the shifted matrix factorizations for this problem.

These results were calculated on a computer with an 8 core Intel Xeon X5550 CPU, with 64 GB of RAM. Performance would likely be improved substantially by using a computer with more RAM, or by running FEAST in parallel across multiple compute nodes; this would make it possible, at the very least, to save and reuse matrix factorizations. Even so, the problem with direct solvers that Table 4.2 highlights will eventually arise regardless of the size and power of the computing system that is being used, provided that one attempts to solve an eigenvalue problem that is large enough.

Ga41As41H72 IFEAST-BiCGSTAB and FEAST-PARDISO Comparison

	Iterations	**Time(s)**
IFEAST-BiCGSTAB	15	4,982
FEAST-PARDISO	10	152,597

Table 4.2. *Iteration counts and total calculation time for using FEAST to calculate the lowest 50 eigenvalues of the Ga41As41H72 eigenvalue problem. Results are shown for IFEAST implemented with BiCGSTAB, using 100 BiCGSTAB iterations at each IFEAST iteration, and for standard FEAST implemented with the PARDISO direct linear system solver. The value of m_0 is 75 and $n_c = 4$ (8 total quadrature points, but I use Hermitian FEAST here, and so i only need to calculate half of the linear systems).*

IFEAST is not the only alternative to standard FEAST. Another natural point of comparison is a Krylov subspace method like Arnoldi[13]. We can also try to solve the Ga41As41H72 problem with the ARPACK software package [49], which implements Arnoldi with implicit restarts. If the maximum subspace dimension for ARPACK is set to be 75 (thus requiring the same amount of memory for basis vector storage, and the same size Rayleigh-Ritz problem, as IFEAST), the total number of required matrix-vector products for IFEAST and ARPACK are

$$\text{IFEAST matvec} = 270{,}000, \quad \text{ARPACK matvec} = 7{,}142.$$

From this perspective, IFEAST appears to require substantially more computational work than does ARPACK. IFEAST, however, is a naturally parallelizable algorithm, whereas ARPACK is inherently sequential. Table 4.3 shows the required number of sequential matrix vector multiplications when IFEAST is run in parallel, using several parallelization schemes. If we solve all of the right hand sides of the IFEAST linear

	All Sequential	**Parallel in n_c**	**Parallel in m_0**	**All Parallel**
IFEAST matvec	270,000	67,500	3,600	900

Table 4.3. *The number of required sequential matrix-vector multiplications for IFEAST to converge on the 50 lowest eigenvalues of Ga41As41H72, for different parallelization schemes. "Parallel in n_c" refers to solving all of the shift linear systems of equations in parallel, and "Parallel in m_0" refers to solving all of the right hand sides for each linear system of equation in parallel.*

systems of equations in parallel, then IFEAST requires fewer sequential matrix-vector products than ARPACK does. If maximum parallelization is used - i.e. all of linear systems of equations are solved in parallel, and all of the right hand sides for each

[13]In practice the Hermitian Lanczos algorithm would usually be more appropriate for a Hermitian problem like Ga41As41H72, because it allows for the use of short recurrences. I use Arnoldi here simply because the highly optimized ARPACK software allows for an easy comparison to a best-in-class implementation of a Krylov algorithm, and I only care about examining the number of required matrix-vector multiplications.

linear system of equation are solved in parallel - then IFEAST requires *substantially* fewer sequential matrix-vector products than ARPACK does.

The number of required matrix-vector products for Arnoldi can be reduced by using a larger subspace dimension, and consequently fewer restarts. Table 4.4 (on page 144) shows the required number of matrix-vector products for ARPACK for different values of the maximum subspace dimension. Although the number of matrix-vector products can be reduced substantially, it can never be made as low as 900, which is the number of sequential matrix-vector products that IFEAST requires when using maximum parallelism.

Subspace Dimension	75	300	2100
ARPACK matvec	7142	3100	2100

Table 4.4. *Total number of required matrix-vector products for ARPACK for different maximum subspace dimensions.*

IFEAST and Arnoldi both calculate approximate eigenvectors by implicitly using polynomial filters; the difference between them lies in how they calculate their respective filters. Arnoldi minimizes the total number of matrix-vector multiplications that are needed to calculate a suitably accurate filter, at the price of having to perform all of its calculations sequentially. IFEAST uses substantially more matrix-vector products in order to calculate a filter, but it does so in such a way that almost all of those matrix-vector products can be done in parallel.

CHAPTER 5

SPECTRAL SLICING WITH CONTOUR INTEGRATION FOR NONLINEAR EIGENVALUE PROBLEMS

The previous chapters in this dissertation discussed variations of the FEAST algorithm for solving linear eigenvalue problems. Solving linear eigenvalue problems by using only matrix-vector multiplication led to a modification of the FEAST contour integral that made it efficient for solving generalized eigenvalue problems, and which allowed for the use of preconditioners. The same modification of the contour integral that allows IFEAST to be used efficiently with generalized eigenvalue problems also allows the FEAST algorithm to be extended to solving *nonlinear* eigenvalue problems [23].

5.1 Nonlinear Eigenvalue Problems

Nonlinear eigenvalue problems consist of finding vectors x and complex scalars λ that satisfy [33, 53, 104]

$$T(\lambda)x = 0, \tag{5.1}$$

where $T(\lambda)$ is an $n \times n$ matrix-valued function called the "residual function". Linear eigenvalue problems $Ax = \lambda Bx$ are the special case of nonlinear eigenvalue problems for which $T(\lambda) = \lambda B - A$. The vector x in Equation (5.1) is a right eigenvector; as with linear eigenvalue problems, there are also left eigenvectors x_L such that

$$x_L^H T(\lambda) = 0. \tag{5.2}$$

Any matrix-valued function $T(\lambda)$ defines an eigenvalue problem, but some classes of functions are more commonly studied than others. Quadratic eigenvalue problems [97] are those where

$$T(\lambda) = \lambda^2 A_2 + \lambda A_1 + A_0, \tag{5.3}$$

so-called because they are quadratic matrix polynomials in λ. This can be generalized to k-degree polynomial eigenvalue problems [51], where

$$T(\lambda) = \sum_{i=0}^{k} \lambda^i A_i. \tag{5.4}$$

There are also analytic eigenvalue problems, wherein $T(\lambda)$ is not a polynomial, but it does have an infinite series expansion [104].

Nonlinear eigenvalue problems have applications that, in many cases, mirror those of their linear counterparts. Where a linear eigenvalue problem can be used to analyze the properties of a first-order time dependent system of equations, a polynomial eigenvalue problem of degree k can be used to analyze the properties of a k^{th}-order time dependent system of equations [97]. Linear eigenvalue problems can be used for dimensionality reduction by calculating the SVD of a data set, which is equivalent to solving a total least squares problem; quadratic eigenvalue problems can be used for dimensionality reduction by solving *regularized* total least squares problems [83]. The resonant states of open quantum systems, in which the particle number can change as particles enter or leave the system, can also be analyzed in terms of nonlinear eigenvalue problems [81, 82].

Nonlinear eigenvalue problems are somewhat more challenging to solve than linear eigenvalue problems are, in part because they violate the expectations and intuitions that are developed from considering linear eigenvalue problems. Whereas linear eigenvalue problems of dimension n can have, at most, n eigenvalues and eigenvectors, nonlinear eigenvalue problems can have many more, potentially an infinite number.

The eigenvectors of a nonlinear eigenvalue problem therefore generally do not form a basis set, orthogonal or otherwise.

There is a biorthogonality relationship between the left and right eigenvectors, as there is with linear eigenvalue problems, but it takes a different form. The left and right eigenvectors are biorthogonal with respect to the scalar product [104]

$$(x_L, x_R)_{nl} = \begin{cases} x_L^H \frac{T(p(x_R))-T(p(x_L))}{p(x_R)-p(x_L)} x_R & p(x_R) \neq p(x_L) \\ x_L^H T'(p(x_R)) x_R & p(x_L) = p(x_L) \end{cases} \tag{5.5}$$

where $T'(\lambda)$ is the derivative of $T(\lambda)$ and $p(x)$ is called the Rayleigh functional. In analogy to the linear eigenvalue problem case, the Rayleigh functional maps vectors to scalars such that eigenvectors are mapped to their corresponding eigenvalues. The Rayleigh functional $p(x)$ is the solution to the equation

$$x^H T(p(x))x = 0. \tag{5.6}$$

For example, for a quadratic eigenvalue problem, the Rayleigh functional is a solution to a quadratic equation.

There are a variety of methods for solving nonlinear eigenvalue problems, many of which are based either on Newton-type iterations or on well-known methods of solving linear eigenvalue problems [2, 3, 8, 35, 41, 66, 68, 92, 103, 108]. Of particular interest for this dissertation are Residual Inverse Iteration [40, 57] and Beyn's method [9], both of which are related to the nonlinear variation of the FEAST algorithm.

5.1.1 Residual Inverse Iteration

For the nonlinear eigenvalue problem, Shift and Invert Iteration takes the form [57, 66]

$$\tilde{x}^{(i+1)} = T(z)^{-1} T'(\lambda^{(i)}) \tilde{x}^{(i)}, \tag{5.7}$$

where $\tilde{x}^{(i)}$ and $\tilde{\lambda}^{(i)}$ are an estimated eigenvector and eigenvalue pair at iteration i. Equation (5.7) is derived by using Newton's method to solve the original nonlinear eigenvalue problem in Equation (5.1). For linear eigenvalue problems Equation (5.7) is exactly equivalent to linear Shift and Invert Iteration.

In order for nonlinear Shift and Invert Iteration to work, the shift z must be set to $z = \tilde{\lambda}^{(i)}$ at every iteration (making it equivalent to Rayleigh Quotient Iteration in the linear case). If the shift z is constant by iteration, as it is with linear Shift and Invert Iteration, then nonlinear Shift and Invert Iteration will fail to converge to a correct solution of the nonlinear eigenvalue problem [57]. Changing the shift z at each iteration is a potential downside to nonlinear Shift and Invert Iteration, because it requires that the matrix $T(z)$ be refactorized each time z changes in order to solve linear systems of equations. The required number of factorizations may ultimately be very high, and they must necessarily be performed sequentially.

Residual Inverse Iteration is an alternative to nonlinear Shift and Invert Iteration that addresses this problem by approximating the derivative $T'(\tilde{\lambda}^{(i)})$ as

$$T'(\tilde{\lambda}^{(i)}) \approx \frac{T(z) - T(\tilde{\lambda}^{(i)})}{z - \tilde{\lambda}^{(i)}}, \tag{5.8}$$

Substituting Equation (5.8) into Equation (5.7) produces

$$\tilde{x}^{(i+1)} = \frac{I - T(z)^{-1}T(\tilde{\lambda}^{(i)})}{z - \tilde{\lambda}^{(i)}}\tilde{x}^{(i)}, \tag{5.9}$$

which is the procedure that defines Residual Inverse Iterations [57]. The denominator $z - \tilde{\lambda}^{(i)}$ is always omitted in practice, since the updated vector $\tilde{x}^{(i+1)}$ is normalized after calculating it.

Unlike nonlinear Shift and Invert Iteration, Residual Inverse Iteration will still converge to the correct solutions of the nonlinear eigenvalue problem in Equation (5.1) when z is set to a constant value near where one expects to find eigenvalues.

When z is constant then iterations of this form converge only linearly, as opposed to quadratically for Equation (5.9) [33], but they have the benefit that only a single matrix factorization needs to be calculated for $T(z)$.

The transition from nonlinear Shift and Invert Iteration to Residual Inverse Iteration with a constant shift is, notably, essentially the same mathematical trick that leads from Rayleigh Quotient Iteration to Simplified Jacobi-Davidson Iteration (see Section 2.3.7 on page 51), from Inexact Shift and Invert Iteration to Generalized Inexact Shift and Invert Iteration (see Section 2.3.6 on page 45), and from Basic IFEAST to Generalized IFEAST (Section 4.3 on page 82). And so, naturally, we will also use it as the basis for Nonlinear FEAST.

5.1.2 Beyn's Method

The solution of nonlinear eigenvalue problems stands to benefit from spectral slicing (see Section 2.4.2 on page 57) in much the same way as the solution of linear eigenvalue problems does. By selectively calculating the eigenvectors whose eigenvalues lie in a specific region in the complex plane, one can efficiently calculate large numbers of eigenpairs in parallel. Spectral slicing also offers the possibility of alleviating additional problems that are unique to nonlinear eigenvalue problems, specifically the difficulty of choosing a good initial guess for the eigenvectors of interest, and the challenge of calculating groups of multiple eigenpairs whose eigenvalues are in close proximity to each other, but without repeatedly converging to the same eigenpair [104].

As with linear eigenvalue problems, the Cauchy integral formula is an attractive choice for a spectral slicing tool. Applying the Cauchy integral formula to the residual function $T(\lambda)$ for a contour $\mathcal{C}$ that encloses m distinct eigenvalues[1] produces [9, 10]

[1]Relations like this also exist for eigenvalues with multiplicity greater than one, but they are more complicated and less easy to compare with the linear case; I consider only eigenvalue problems of multiplicity one for the sake of simplicity.

$$\frac{1}{2\pi i}\oint_{\mathcal{C}} z^k T^{-1}(z)\, dz = X_R \Lambda^k X_L^H, \tag{5.10}$$

where X_R and X_L are $n \times m$ matrices whose column vectors are the m right and left eigenvectors with eigenvalues inside $\mathcal{C}$, and Λ is a diagonal matrix whose diagonal entries are the corresponding eigenvalues.

When the integral in Equation (5.10) is evaluated exactly then the result, $X_R \Lambda^k X_L^H$, projects any vector on to a subspace spanned by only the right eigenvectors X_R that correspond to eigenvalues that are inside $\mathcal{C}$. It is thus tempting to say that the right hand side of Equation (5.10) is a spectral projector for the eigenvectors of interest when $k = 0$, which would allow us to use the standard FEAST algorithm for nonlinear eigenvalue problems directly, but this is incorrect. The right and left eigenvectors X_R and X_L are not biorthogonal, meaning that $X_L^H X_R \neq I$, and so the diagonal values of Λ^k do not act as a filter for a vector that is expressed as a linear combination of the column vectors of X_R. In fact, the right eigenvectors as a whole do not form a basis set, and the approximation of the integral in (5.10) with a quadrature sum does not form an approximate spectral projector, as it does in the linear case. For these reasons the filtering approach of the linear FEAST algorithm does not work for nonlinear eigenvalue problems.

It is possible, none the less, to use the contour integration in Equation (5.10) to find accurate approximations of the eigenvalues in $\mathcal{C}$ and corresponding eigenvectors [2, 9, 92, 108]. A simple and effective approach is to use Beyn's method [9]. Rather than taking an iterative filtering approach, as the linear FEAST algorithm does, Beyn's method uses a single, high-precision numerical integration to approximate Equation (5.10) for both $k = 0$ and $k = 1$. The results of these integrations are used to form a reduced-dimension *linear* eigenvalue problem whose solutions are the nonlinear eigenvalues and eigenvectors of interest. Beyn's method is summarized in Algorithm 9 (on page 151).

Algorithm 9 Beyn's Method for Nonlinear Eigenvalue Problems

Inputs:

- $n \times n$ matrix-valued function $T(\lambda)$,
- Contour $\mathcal{C}$ in the complex plane that encloses the search region for eigenvalues,
- Collection of n_c quadrature weights and shifts (ω_k, z_k) for performing integrations over $\mathcal{C}$,
- Overestimate m_0 for the number of eigenvalues that are inside $\mathcal{C}$,
- $n \times m_0$ matrix $\tilde{X}^{(0)}$ whose column vectors are (possibly random) initial guesses for the eigenvectors of $T(\lambda)$ whose, eigenvalues are inside $\mathcal{C}$.

Do:

1. Calculate the numerical integrations

$$Q_0 = \sum_{k=1}^{n_c} \omega_k T(z_k)^{-1} \tilde{X}^{(0)} \tag{5.11}$$

$$Q_1 = \sum_{k=1}^{n_c} \omega_k z_k T(z_k)^{-1} \tilde{X}^{(0)} \tag{5.12}$$

   by solving n_c linear systems of equations.
2. Calculate the Singular Value Decomposition $Q_0 = USV^H$
3. Form the $m_0 \times m_0$ matrix $A = U^H Q_1 V S^{-1}$
4. Calculate the eigenvalue decompsition $A = X_a \Lambda_a X_a^{-1}$
5. Form the approximate eigenvalues and eigenvectors of $T(\lambda)$

$$\tilde{X} = U X_a \tag{5.13}$$

$$\tilde{\Lambda} = \Lambda_a \tag{5.14}$$

Outputs: diagonal matrix $\tilde{\Lambda}$ of approximations for the eigenvalues inside $\mathcal{C}$, and approximations for the corresponding eigenvectors $\tilde{X}$.

The summary of Beyn's method that is presented in Algorithm 9 is a simplified version of the original in [9]. In his original paper, Beyn suggests an implementation of the algorithm that does not require m_0 to be an overestimate for the number of eigenvalues inside $\mathcal{C}$. Instead, the number of column vectors in $\tilde{X}^{(0)}$ is successively increased from some initial value until the rank of the matrix Q_0 begins to drop below its number of column vectors, based on some user-set tolerance for its singular values.

Beyn's method has the advantages that we would hope to find in a spectral slicing algorithm: it can simultaneously calculate multiple eigenvectors whose eigenvalues

are in specific regions of the complex plane, using a random initial guess. Its most expensive computational task is the solution of n_c linear systems of equations in **Step 1** of Algorithm 9, and these linear systems of equations can all be calculated in parallel if a sufficient amount of parallel computing power is available.

Beyn's method has the potential disadvantage, however, that the accuracy of its solutions depends entirely on the accuracy with which it approximates the Cauchy integral in Equation (5.10). The number of quadrature points n_c that is necessary for achieving sufficiently low eigenvector residual norms can potentially be very large. A truly efficient implementation must use an adaptive integration quadrature method, so that if one uses a value of n_c that is not large enough then the integration can be updated without having wasted all of the previous computational work. Perhaps most importantly, the linear systems of equations that need to be solved in order to implement Step 1 of Algorithm 9 must be solved to high accuracy in order to ensure that the contour integration is sufficiently accurate. This limits Beyn's method to solving small or medium-sized nonlinear eigenvalue problems, where the linear systems of equations can be solved exactly by using matrix factorization methods.

5.2 FEAST for Nonlinear Eigenvalue Problems

The framework of the FEAST algorithm offers an attractive alternative for using spectral slicing with nonlinear eigenvalue problems. The FEAST algorithm, too, takes advantage of the properties of Cauchy integrals, which allows it to simultaneously calculate multiple eigenvectors whose eigenvalues lie in a specific region in the complex plane using a random initial guess. FEAST is also a naturally iterative algorithm; if the eigenvectors that are produced by performing a single numerical integration are not sufficiently accurate, then they can be systematically refined by performing yet another numerical integration using exactly the same quadrature weights and shifts as the first integration. FEAST can thus save and reuse matrix factorizations. Because

it is naturally iterative, it can also be used when the linear systems of equations can only be solved approximately, making it an appropriate algorithm for solving large eigenvalue problems.

As mentioned in Section 5.1.2, the Cauchy integral for nonlinear problems is

$$\frac{1}{2\pi i}\oint_{\mathcal{C}} T^{-1}(z)\, dz = X_R X_L^H, \tag{5.15}$$

where X_R and X_L are $n \times m$ matrices whose column vectors are the eigenvectors corresponding to the m distinct eigenvalues inside the closed contour $\mathcal{C}$ in the complex plane. An approximation of this integral can not be used as a spectral projector, unlike the linear case, and so we can not simply replace the linear resolvent $(zI - A)^{-1}$ from the linear FEAST algorithm with $T(z)^{-1}$ to produce a nonlinear FEAST algorithm.

Instead, we can arrive at a nonlinear version of the FEAST algorithm by noting that the linear FEAST algorithm can be interpreted as a generalization of Shift and Invert Iteration that uses multiple shifts. The nonlinear version of Shift and Invert Iteration uses the eigenvector update rule

$$\tilde{x}^{(i+1)} = T(z)^{-1}T'(\lambda^{(i)})\tilde{x}^{(i)}, \tag{5.16}$$

We would then expect a nonlinear FEAST contour integral (for a single vector) to be something like

$$q = \frac{1}{2\pi i}\oint_{\mathcal{C}} T(z)^{-1}T'(\lambda^{(i)})\tilde{x}^{(i)} dz, \tag{5.17}$$

where the integration contour $\mathcal{C}$ encloses an eigenvalue of interest.

In practice, however, Equation (5.17) does not work. Shift and Invert Iteration fails to converge to a correct solution when the shift z is a constant, and the integral in Equation (5.17) simply amounts to Shift and Invert Iteration with an infinite number of constant shifts.

We can derive a viable contour integral for a nonlinear version of FEAST by altering Equation (5.17) using the same approximation that is used to derive Residual Inverse Iteration from nonlinear Shift and Invert Iteration, i.e.

$$T'(\tilde{\lambda}^{(i)}) \approx \frac{T(z) - T(\tilde{\lambda}^{(i)})}{z - \tilde{\lambda}^{(i)}}. \tag{5.18}$$

Inserting this into Equation (5.17) produces

$$q = \frac{1}{2\pi i} \oint_{\mathcal{C}} \left(I - T(z)^{-1} T(\tilde{\lambda}^{(i)}) \right) \tilde{x}^{(i)} \frac{1}{z - \tilde{\lambda}^{(i)}} \, dz \tag{5.19}$$

The block version of this integral is

$$Q = \frac{1}{2\pi i} \oint_{\mathcal{C}} \left(I - T(z)^{-1} T(\tilde{\Lambda}^{(i)}, \tilde{X}^{(i)}) \right) (zI - \tilde{\Lambda}^{(i)})^{-1} \, dz \tag{5.20}$$

where I use the notation $T(\tilde{\Lambda}^{(i)}, \tilde{X}^{(i)})$ is to indicate the residual function $\tilde{T(\lambda)}$ applied to each column of $\tilde{X}^{(i)}$ individually at the corresponding approximate eigenvalue $\tilde{\lambda}^{(i)}$.

For linear eigenvalue problems Equation (5.20) is exactly equivalent to the integral that is used in the standard FEAST algorithm, and indeed Equation (5.20) is the form of the Cauchy integral that is used for implementing Generalized IFEAST (see Section 4.3 on page 82). As we will see in the following sections, it produces exactly the behavior that we would expect of the FEAST algorithm when applied to nonlinear eigenvalue problems as well.

The Nonlinear FEAST algorithm is summarized in Algorithm 10. It is largely the same as the linear Generalized Inexact FEAST algorithm, with the contour integration having been generalized to accommodate nonlinear eigenvalue problems of any form. There is one important difference between the two, however, which is the Rayleigh-Ritz step.

Algorithm 10 NLFEAST for $T(\lambda)x = 0$

Inputs:

- Matrix function $T(\lambda) \in \mathbb{C}^{n\times n}, \quad \lambda \in \mathcal{C}$
- Initial (possibly random) guess $\tilde{X}^{(0)} \in \mathbb{C}^{n\times m_0}$ for the search subspace (initial set of estimated eigenvectors)
- Closed contour $\mathcal{C}$, inside of which fewer than m_0 eigenvalues are expected to be found
- Set of n_c quadrature nodes and weights (z_j, ω_j) for performing numerical integration over the contour $\mathcal{C}$
- Stopping tolerance ϵ for eigenvector residuals

Step 0. Set the search subspace $Q = \tilde{X}^{(0)}$, orthonormalize column vectors of Q

For each iteration:

Step 1. (Rayleigh-Ritz step) Solve the projected nonlinear eigenvalue problem

$$Q^H T(\lambda) Q y = 0, \tag{5.21}$$

for the approximate eigenpairs (λ, Qy).

Step 2. Select the m_0 approximate eigenpairs (λ_i, Qy_i) whose eigenvalues λ are closest to the interior of the contour $\mathcal{C}$, and store these as Λ, and X, where the columns of X are Qy_i and the diagonal entries of Λ are λ_i, i.e.

$$\Lambda = \mathrm{diag}(\lambda_1, \lambda_2, ..., \lambda_{m_0}), \quad X = [Qy_1, Qy_2, ..., Qy_{m_0}]. \tag{5.22}$$

Step 3. If $||T(\lambda_i)Qy_i|| \leq \epsilon$ for all λ_i inside $\mathcal{C}$ **STOP** and return X and Λ; otherwise continue.

Step 4. Update the search subspace by performing the numerical integration

$$Q = \frac{1}{2\pi i} \oint_{\mathcal{C}} \left(X - T^{-1}(z)T(X,\Lambda)\right)(zI - \Lambda)^{-1} dz \tag{5.23}$$

by using a quadrature rule and solving linear systems:

$$Q = \sum_{j=1}^{n_c} \omega_j \left(X - T^{-1}(z_j)T(X,\Lambda)\right)(z_j I - \Lambda)^{-1}. \tag{5.24}$$

Step 5. Orthonormalize the column vectors of the new search subspace Q (using e.g. the QR decomposition), and go to **Step 1.**

In the linear FEAST algorithm - either standard FEAST or IFEAST - eigenvectors and eigenvalues are approximated by solving a smaller eigenvalue problem of dimension m_0, which provides m_0 new estimates for eigenvalues and eigenvectors. If the residuals of these approximate eigenpairs are too large then they are used as the initial guess for another FEAST iteration.

The NLFEAST algorithm, in turn, approximates eigenvalues and eigenvectors by solving a *nonlinear* eigenvalue problem of dimension m_0. A nonlinear eigenvalue problem of dimension m_0, however, will generally have more than m_0 solutions. One must then select only m_0 of the resulting solutions to be used as initial guesses for the next NLFEAST iteration. Selecting these solutions incorrectly will prevent NLFEAST from converging to the desired eigenpairs.

The following subsections describe two effective and straight-forward methods for solving the projected nonlinear eigenvalue problem of NLFEAST and selecting the correct m_0 solutions to return for the next iteration.

5.2.1 Polynomial Linearization

The first method is particular to polynomial eigenvalue problems, and it consists of using linearization in order to calculate all of the solutions of the projected polynomial eigenvalue problem. The linearization of an n dimensional polynomial eigenvalue problem of degree k is a linear $(n \times k)-$dimensional eigenvalue problem that has the same eigenvalues as the original matrix polynomial, and whose eigenvectors are related to the eigenvectors of the matrix polynomial [29]. For example, the linearization of the quadratic eigenvalue problem

$$(\lambda^2 A_2 + \lambda A_1 + A_0)x = 0 \tag{5.25}$$

is the linear, generalized eigenvalue problem

$$\left(\lambda \begin{bmatrix} A_2 & 0 \\ 0 & I \end{bmatrix} + \begin{bmatrix} A_1 & A_0 \\ -I & 0 \end{bmatrix} \right) y = 0. \tag{5.26}$$

The eigenvalues λ that are solutions to Equation (5.25) are the same as the eigenvalues λ that are solutions to Equation (5.26). The eigenvector solutions y of Equation (5.26) are related to the eigenvector solutions x of Equation (5.25) (for same eigenvalue λ) by

$$y = \begin{bmatrix} \lambda x \\ x \end{bmatrix}. \tag{5.27}$$

Thus, when using NLFEAST to solve polynomial eigenvalue problems, we can directly solve the projected polynomial eigenvalue problem of the Rayleigh-Ritz step by using a linearization like Equation (5.26).

Solving a dimension m_0, degree k polynomial eigenvalue problem by using linearization produces $m_0 \times k$ solutions, of which we need to select m_0 to return for the next iteration. An effective heuristic for selecting the desired solutions from the linearization is to select the m_0 eigenpairs whose eigenvalues are closest to being inside of the search contour. For a circular contour this simply means selecting the m_0 eigenpairs whose eigenvalues are closest to the center of the contour. It is worth emphasizing that this procedure does *not* involve considering the eigenvector residuals of the solutions that are produced by solving the projected polynomial eigenvalue problem. If one selects the solutions with the lowest eigenvector residuals, then the approximate solutions to the polynomial eigenvalue problem may converge to eigenpairs whose eigenvalues are outside of the search contour, or they may not converge at all.

5.2.2 Beyn's Method

Using linearization for solving the projected nonlinear eigenvalue problem in the Rayleigh-Ritz step of NLFEAST will only work for polynomial eigenvalue problems

and, moreover, it will only work when the subspace dimension m_0 and the polynomial degree k are such that an $m_0 \times k$ dimension linear eigenvalue problem is not too large to solve. A different approach is needed for general nonlinear eigenvalue problems.

In principle, any method for solving general nonlinear eigenvalue problems can be used for solving the projected nonlinear eigenvalue problems in the NLFEAST Rayleigh-Ritz step. Most methods of solving nonlinear eigenvalue problems appear to be inappropriate or difficult to use for our purposes, though, because many of them operate on only a single vector at a time, are specialized for certain classes of problems, and/or can not be used to identify eigenvalues in specific regions in the complex plane. Instead, a simple and obvious choice for solving the projected nonlinear eigenvalue problems of NLFEAST is to simply to use Beyn's method, which can naturally be used to calculate exactly m_0 eigenpairs whose eigenvalues are near or inside of a given search contour in the complex plane.

Beyn's method synergizes naturally with the NLFEAST algorithm. The potential drawback of Beyn's method is that a potentially large number of matrix factorizations may be required for it to work well. This is not an obstacle when the dimension of the nonlinear eigenvalue problem under consideration is very small, as it ought to be for the projected nonlinear eigenvalue problems of NLFEAST. By using NLFEAST as the outer iteration, and Beyn's method as the inner iteration, we can combine the simplicity and effectiveness of Beyn's method with the efficiency of NLFEAST.

5.3 Examples

This section describes the solution of several example problems from the Nonlinear Eigenvalue Problem Collection [7] in order to demonstrate the properties of NLFEAST, and specifically to demonstrate that its behavior largely matches that of the linear FEAST algorithm. Three kinds of examples are considered that represent a

range of prospective nonlinear eigenvalue problems: a quadratic polynomial problem, and quartic polynomial problem, and a general nonlinear problem.

5.3.1 Rail Track Oscillations

The oscillations of long lengths of railroad track can be modeled by quadratic eigenvalue problem of the form [7, 47]

$$T(\lambda) = \lambda^2 I + \lambda(I + A^2) + A^2 + A + I, \tag{5.28}$$

where A is an $n \times n$ circulant matrix with first row

$$[-2,\ 1,\ 0,\ \ldots,\ 0,\ 1]. \tag{5.29}$$

For any dimension n, the eigenvalues of the quadratic eigenvalue problem (5.28) are given by the solutions to the quadratic equation

$$\lambda^2 + \lambda(1 + \mu_k^2) + (1 + \mu_k + \mu_k^2) = 0, \tag{5.30}$$

where $\mu_k = -4\sin^2((k-1)\pi/n)$, $1 \leq k \leq n$ are the eigenvalues of the matrix A. The eigenvalues occur in complex conjugate pairs, and as a result the real eigenvalues (of which there are many) have multiplicity 2.

This example provides a convenient demonstration of convergence for larger problem sizes. Let us consider the case when the dimension of A is $n = 50,000$, and calculate the 250 eigenvectors corresponding to eigenvalues in the interval $[-7.1192, -6.9650]$ by using NLFEAST with a circular contour centered at -7.0421 with radius 0.0771. The locations of the eigenvalues and the integration contour are illustrated in Figure 5.1.

Figure 5.2 shows NLFEAST convergence trajectories for various values of n_c and m_0. These results are calculated by implementing NLFEAST using linearization for

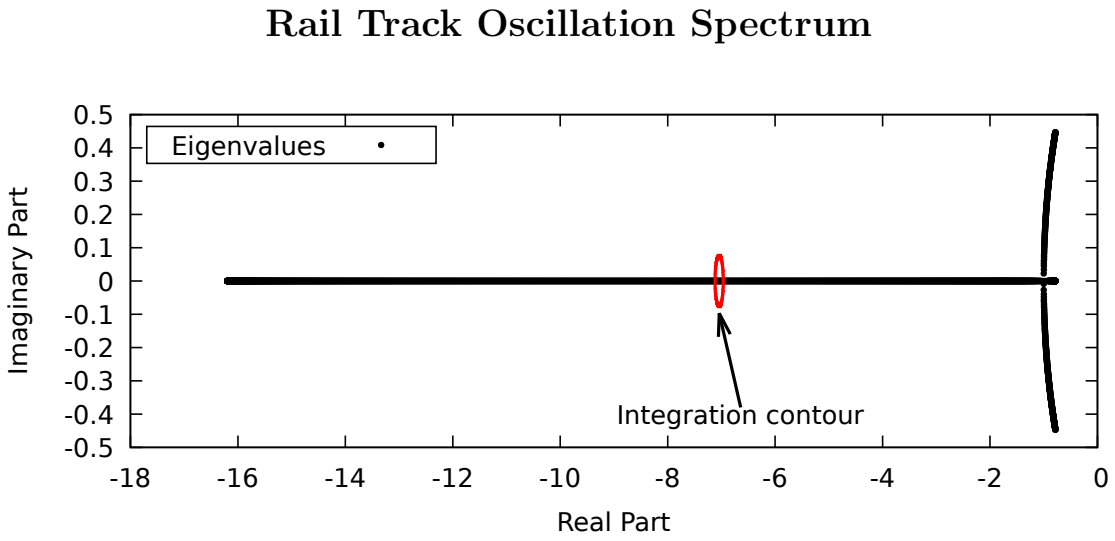


Figure 5.1. *Eigenvalue locations and search contour for the Rail Track Oscillation problem.*

the Rayleigh-Ritz step. These results show that, as with linear FEAST, increasing either the number of integration quadrature points or the dimension of the subspace improves the rate of convergence. The reported NLFEAST residual is the largest residual for the approximate eigenvectors associated with eigenvalues located inside of the integration contour, excluding spurious eigenvalues. A few spurious eigenvalues are excluded by only considering eigenvector residuals that are below 10^{-2} to be inside of the integration contour. In all cases this results in the correct number of eigenvalues being identified inside of the contour.

As a point of comparison we can also use Beyn's method to calculate the same 250 eigenvalues using the same integration contour. Table 5.1 shows the final eigenvector residuals for the eigenvectors corresponding to eigenvalues inside of the integration contour, for various subspace dimensions and various numbers of integration quadrature points. As with the NLFEAST example, spurious eigenvalues are excluded by considering only eigenvectors with residuals below 10^{-2} as being correct eigenvectors, rather than spurious ones. This means that, when $n_c = 32$ and $m_0 = 500$ and

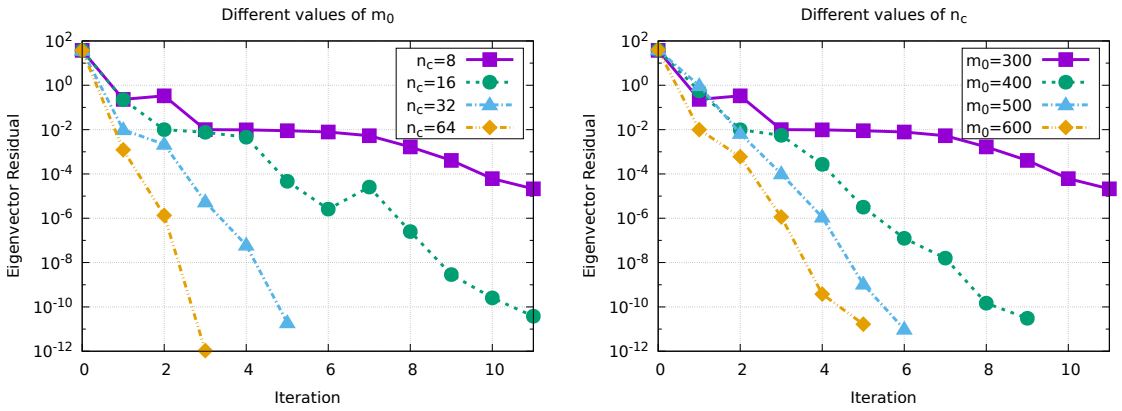


Figure 5.2. *Eigenvector residual versus FEAST iteration for the Rail Track Oscillation problem, for several values of n_c and m_0. The left plot shows results for various values of m_0 with $n_c = 8$, and the right plot shows results for various values of n_c with $m_0 = 300$.*

the eigenvector residual is reported as 9.4×10^{-1}, none of the 500 eigenvectors have a residual below 10^{-2}, and thus all of the approximate eigenvectors are considered to be incorrect. All of the results in Table 5.1 with residuals below 10^{-2}, however, identified the correct number of eigenvalues inside of the contour.

Beyn's Method Residuals for Rail Oscillations

	$n_c = 8$	$n_c = 32$	$n_c = 64$	$n_c = 128$
$m_0 = 300$	2.4e-1	1.0e-2	3.8e-3	4.2e-9
$m_0 = 500$	9.4e-1	3.8e-8	2.5e-12	6.9e-12

Table 5.1. *Final eigenvector residuals when using Beyn's method to calculate the 250 eigenvalues in the interval $(-7.1192, -6.9650)$ for Equation (5.28), with $n = 50,000$, subspace dimension m_0 and number of quadrature points n_c.*

The results in Table 5.1 show that Beyn's method and NLFEAST appear to require similar amounts of computational work in order to calculate a solution. For $m_0 = 500$ and $n_c = 8$, for example, NLFEAST requires 6 iterations to converge to approximately 10^{-11}, which amounts to a total of $500 \times 8 \times 6 = 24,000$ linear system solutions with a single right hand side. Beyn's method, for $m_0 = 500$, requires

$n_c = 64$ quadrature points in order to converge to an eigenvector residual of 10^{-12}, which corresponds to a total of $500 \times 8 \times 8 = 28,500$ linear system solutions with a single right hand side.

The difference between the two lies in how the computational work is done. Beyn's method can, in principle, solve all of its associated linear systems of equations in parallel, where as NLFEAST can solve at most $m_0 \times n_c$ linear systems of equations (each with one right hand side) in parallel. The increased parallelism of Beyn's method comes at the price of having to calculate a larger number of matrix factorizations; in the aforementioned example, Beyn's method required 64 matrix factorizations, whereas NLFEAST required 8. The implicit assumption in this comparison is that an adequate number of quadrature points has been selected for using Beyn's method; in practice Beyn's method should use an adaptive integration scheme in order to be able to increase the number of quadrature points when the integration accuracy is inadequate, whereas NLFEAST can simply perform additional iterations to refine an unconverged solution.

5.3.2 Butterfly Problem

As an example of a polynomial eigenvalue problem with degree larger than two, we can consider the following quartic problem

$$P(\lambda)x = (\lambda^4 A_4 + \lambda^3 A_3 + \lambda^2 A_2 + \lambda A_1 + A_0)x = 0.$$

Eigenvalue problems of this form can come from, for example, discretizations of the Orr-Sommerfeld equation [12, 96]. The Orr-Sommerfeld equation arises from a linearization of the incompressible Navier-Stokes equation in which the perturbations of the pressure and velocity are assumed to be periodic in time.

To illustrate the behavior of the nonlinear FEAST algorithm, we can solve a simple example of a quartic eigenvalue problem provided by the NLEVP collection [7]:

the *butterfly* problem (so-called because the distribution of its eigenvalues in the complex plane resembles the shape of a butterfly). The *butterfly* problem is a 64×64 structured quartic matrix pencil with 256 eigenvalues, the construction of which is described in [52]. To solve this problem I use the NLFEAST algorithm to calculate the eigenvalues that are located inside of some arbitrarily chosen region $\mathcal{C}$ in the complex plane, with Beyn's method used for the Rayleigh-Ritz step. This problem is illustrated in Figure 5.3. I calculate 13 eigenvalues inside of the indicated region by

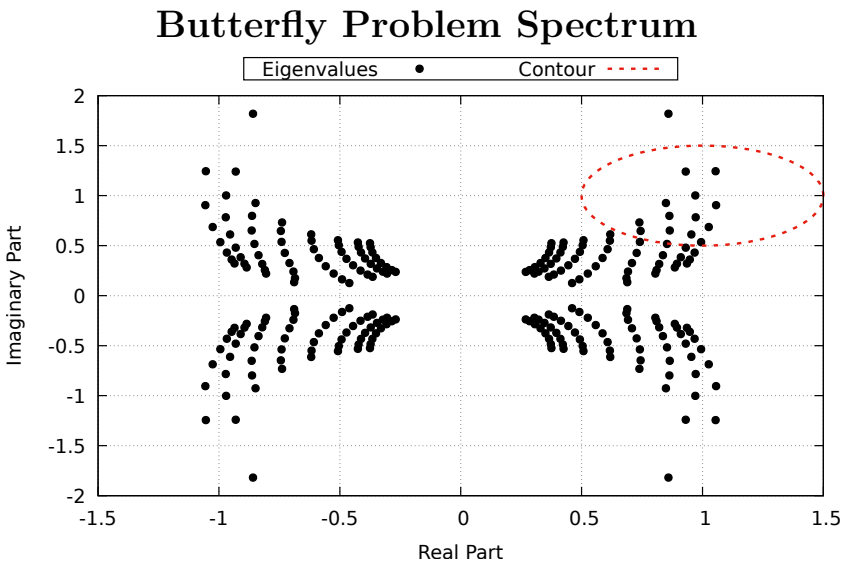


Figure 5.3. *Eigenvalue locations and search contour for the butterfly problem.*

using a subspace of dimension $m_0 = 15$, using several different numbers of quadrature nodes n_c. The largest (at each iteration) eigenvector residual associated with the eigenvectors whose eigenvalues are inside the search region $\mathcal{C}$ is plotted in Figure 5.4. Using $n_c = 8$ quadrature nodes, NLFEAST does not converge at all. Steady convergence can be achieved by using $n_c = 32$ quadrature nodes, and the rate of convergence increases with increasing values of n_c. For $n_c = 128$, convergence to the desired tolerance of 10^{-10} occurs in only five (5) NLFEAST iterations. Because the linear system for each individual quadrature node is independent of the linear systems

associated with all the other quadrature nodes, the rate of convergence of NLFEAST can be systematically improved by using additional parallel processing power to solve a larger number of linear systems simultaneously in parallel.

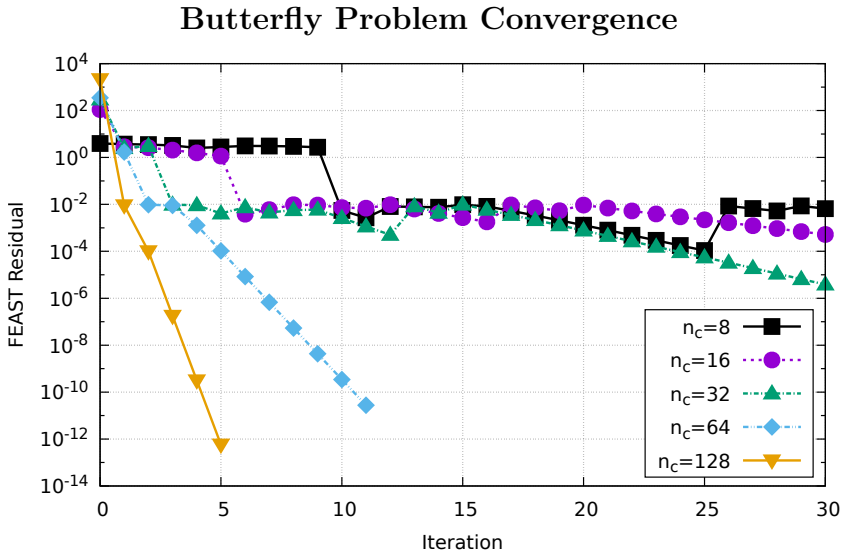


Figure 5.4. *Eigenvector residual versus iteration number for several values of n_c when applying NLFEAST to the butterfly problem.*

Figure 5.5 shows the NLFEAST-estimated eigenvalues for the experiments from Figure 5.4 that use $n_c = 8$ and $n_c = 32$ quadrature points in the numerical integration. The $n_c = 8$ case is not able to converge because the integration is not sufficiently accurate to achieve convergence of the two eigenvalues that are well-separated from the main cluster of eigenvalues; using $n_c = 32$ allows NLFEAST to converge for all of eigenvalues inside the search region $\mathcal{C}$.

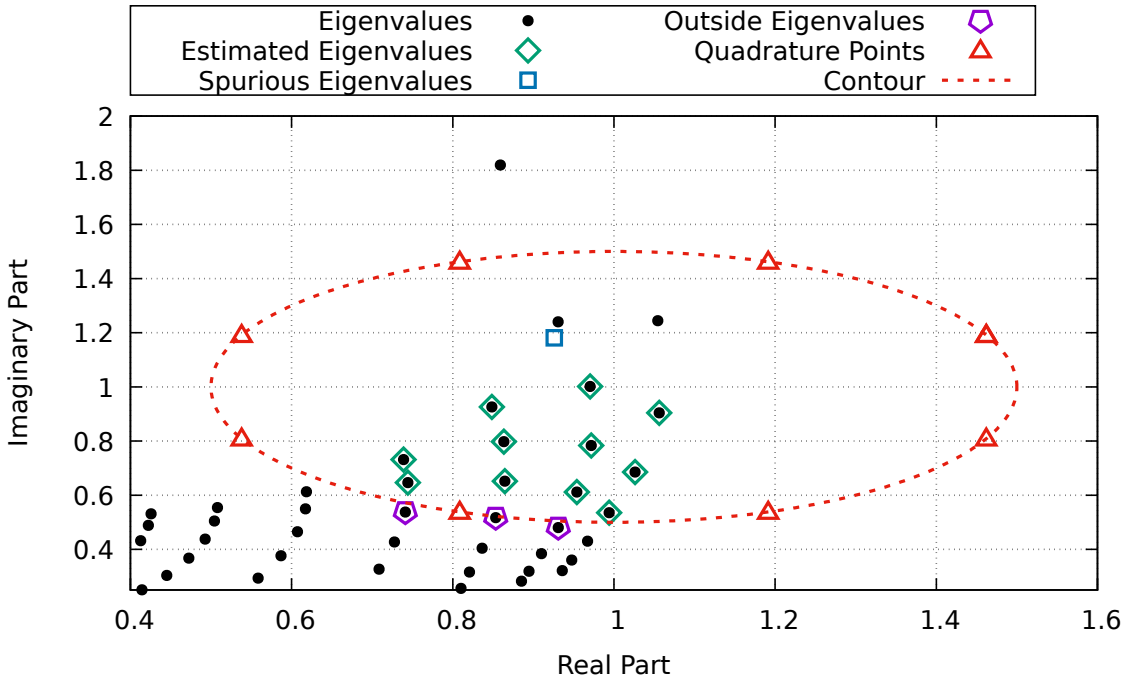
5.3.3 Hadeler Problem

As an example of a general nonlinear eigenvalue problem, we can consider the "Hadeler" problem [7, 36, 66]:

$$T(\lambda) = (e^{\lambda} - 1)B_1 + \lambda^2 B_2 - B_0, \tag{5.31}$$

Butterfly Problem Results

Results for n_c=8



Results for n_c=32

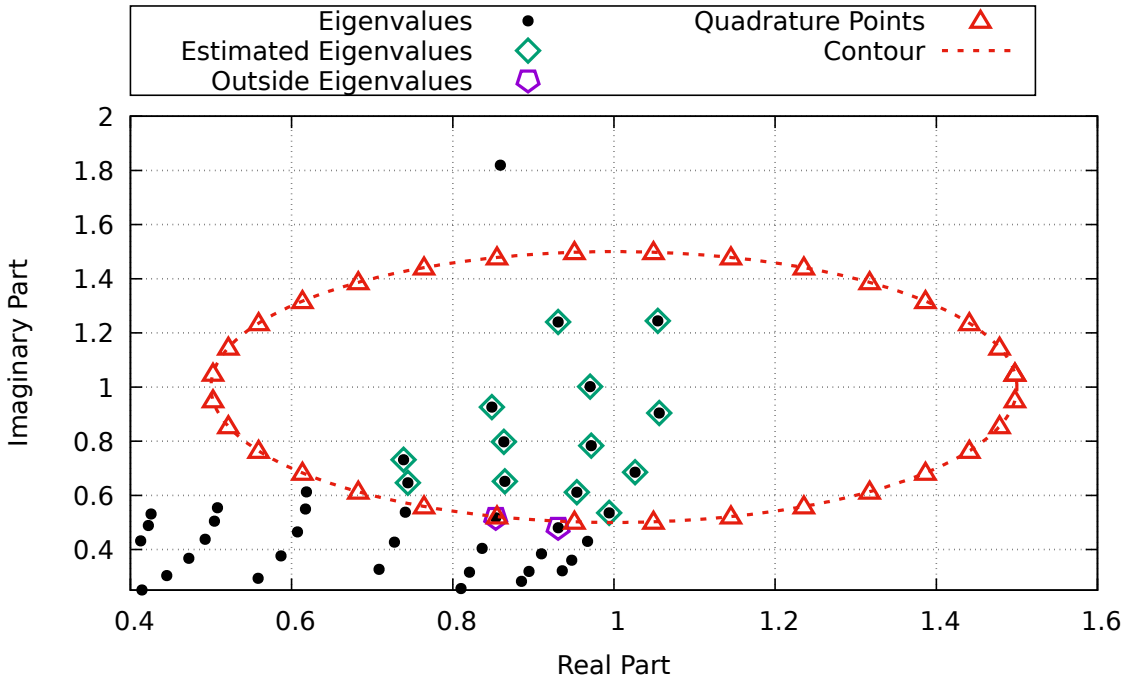


Figure 5.5. *Locations of calculated approximate eigenvalues for the butterfly problem, for two different values of n_c. Correct eigenvalues are distinguished from spurious ones by different plot markers. Spurious eigenvalues indicate eigenvalues that have not converged below a "bare minimum" threshold, which in this case is 10^{-2}.*

with the matrix elements of B_0, B_1, and B_2 being

$$B_0 = b_0 I, \quad B_1 = (b_{jk}^{(1)}), \quad B_2 = (b_{jk}^{(2)}) \tag{5.32}$$

$$b_{jk}^{(1)} = (n + 1 - \max(j,k))jk, \quad b_{jk}^{(2)} = n\delta_{jk} + 1/(j+k), \tag{5.33}$$

where n is the dimension of the eigenvalue problem and b_0 is a parameter that we set as $b_0 = 100$ (following reference [66]). For this example I will set $n = 200$.

The eigenvalues of (5.31) are plotted in Figure 5.6. All of the eigenvalues are real; there are n eigenvalues less than zero, and n eigenvalues greater than zero. I calculate 5 eigenvalues in an arbitrarily-chosen region on the right half of the spectrum, using NLFEAST with Beyn's algorithm for performing the Rayleigh-Ritz step. Figure 5.7 shows plots of the eigenvector residual versus the NLFEAST iteration number, for various values of the parameters n_c and m_0. Figure 5.8 shows the search contour and resulting estimated eigenvalues for $m_0 = 10$ and $n_c = 8$.

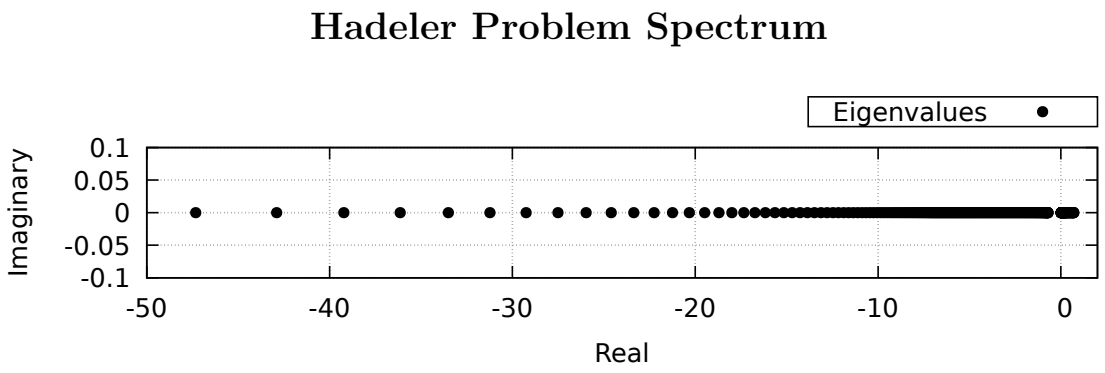


Figure 5.6. *Eigenvalue spectrum for the Hadeler problem.*

As with the previous polynomial eigenvalue problem examples, and as with the linear variation of FEAST, improving the contour integration accuracy by increasing n_c improves the rate of convergence of the NLFEAST algorithm for the Hadeler

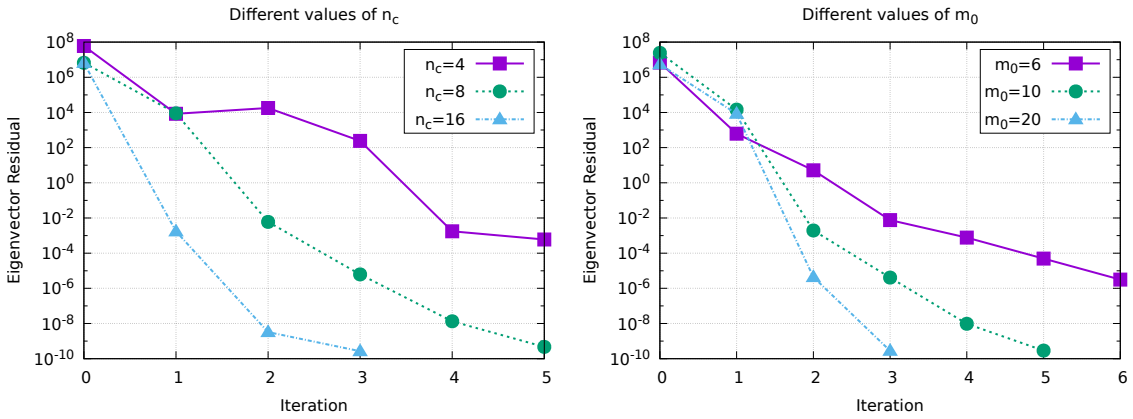


Figure 5.7. *Eigenvector residual versus iteration number for solving the Hadeler problem with NLFEAST. Five eigenvalues in the middle of the left half of the spectrum are calculated. The left plot shows results for several values of n_c, with $m_0 = 10$. The right plot shows results for several values of m_0, with $n_c = 8$.*

problem. It is much less sensitive to changing the dimension of the subspace m_0 however; increase in m_0 provide only modest improvements in the rate of convergence of the NLFEAST algorithm.

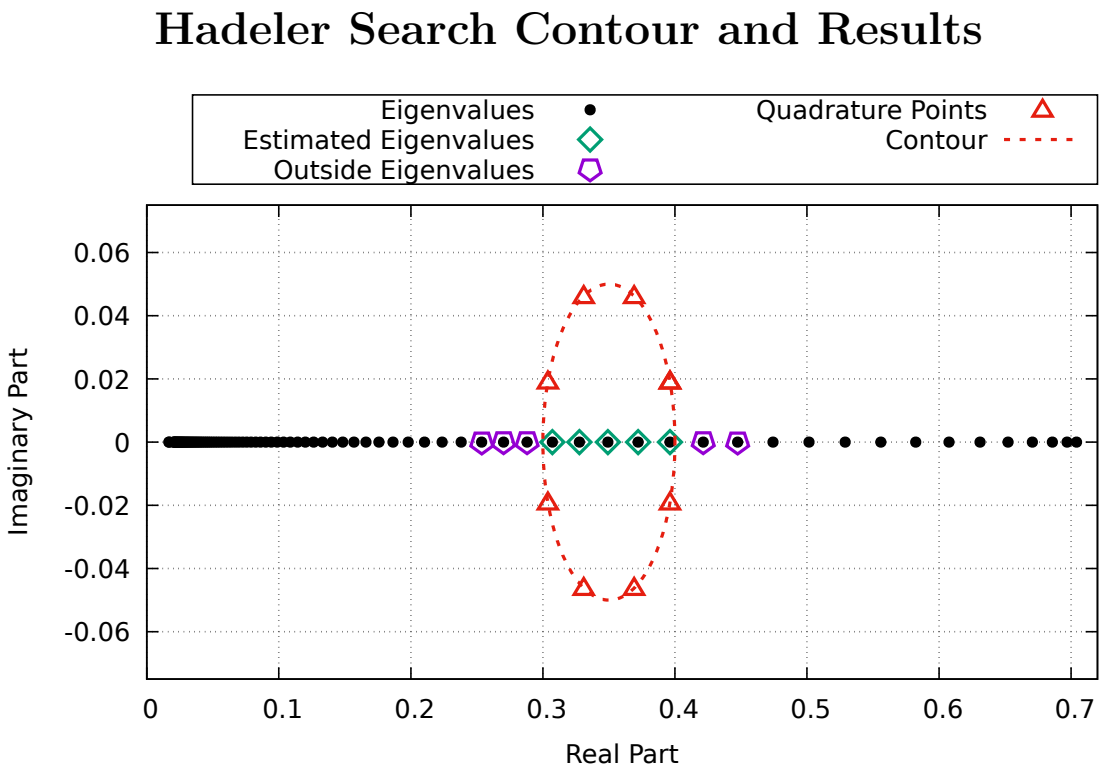


Figure 5.8. *Search contour and calculated eigenvalues for the Hadeler problem.*

CHAPTER 6

CONCLUSION

This dissertation describes modifications of the standard FEAST eigenvalue algorithm that allow it to be applied to new and more challenging problem domains. The IFEAST algorithm is a variation of FEAST that can be used for solving linear eigenvalue problems when it is inefficient or impossible to directly solve linear systems of equations. The NLFEAST algorithm is a variation of FEAST that can be applied to general, nonlinear eigenvalue problems. These two variations of FEAST are, perhaps surprisingly, fundamentally the same algorithm, specialized for particular use cases.

The theory behind IFEAST has been reasonably well-developed. We can show that IFEAST can be expected to converge linearly, and there are mathematical justifications for believing that it is an efficient method for solving both standard and generalized linear eigenvalue problems. For the standard eigenvalue problem we can further interpret the behavior of IFEAST in terms of the action of polynomial filters, which clarifies much of its behavior. IFEAST is less easily parallelized than the standard FEAST algorithm is, and this fact is largely due to the natural limitations of using polynomial filters for solving eigenvalue problems. Even so, IFEAST is much more naturally parallelizable than conventional Krylov subspace-based eigenvalue algorithms are; if enough parallel computing power is available then this fact can be used to solve eigenvalue problems more quickly than would otherwise be possible with many standard algorithms.

It seems likely that there is not a substantial amount of additional room for further improvement in the performance of eigenvalue solving algorithms that operate in

the mode of IFEAST. The restriction that only matrix-vector multiplication should be used as the primary computational tool is a severe one, and it places obvious and inescapable limitations on the effectiveness of any algorithm: one is necessarily limited to what can be accomplished with polynomial filters. Any highly effective algorithm for solving eigenvalue problems under these constraints will necessarily resemble Jacobi-Davidson, IFEAST, or direct polynomial filtering, combined (perhaps) with some kind of additional augmented subspace method. This serves to reinforce a truism that is frequently mentioned in textbooks, which is that one's effort is better-spent on developing effective preconditioners, rather than exhaustively testing algorithms to find the one that is most effective for a specific problem at hand. Effective preconditioners are the only way to escape the limitations imposed by polynomial-based algorithms[1].

The theory behind NLFEAST is much less well-developed. This is also true, to a certain extent, for Generalized IFEAST; the Generalized IFEAST algorithm can not be easily interpreted in terms of polynomial filters when it is applied to generalized eigenvalue problems, and in that situation it would seem to have more in common with the nonlinear eigenvalue problem case. Although we have sound justifications for the particular form that the NLFEAST algorithm takes, as well as numerical results to back it up, a thorough understanding of how the algorithm works is still lacking. Of particular interest is the issue of why the NLFEAST contour integral appears to be so effective. Residual Inverse Iteration, which is a very similar algorithm, has a satisfying interpretation in terms of Newton methods, but that logic does not seem to extend to using multiple shifts or contour integration in an obvious way. A good explanation for why contour integration helps NLFEAST to converge quickly would

[1]One might imagine using some sort of more exotic method of computation, like neural networks, for solving eigenvalue problems, but the take-away lesson is still the same: a neural network that is trained to solve a particular kind of eigenvalue problem ultimately amounts to a sophisticated preconditioner.

presumably offer a new perspective regarding why the original FEAST algorithm works, and would also help to clarify why the Generalized IFEAST algorithm is so effective for generalized eigenvalue problems. These unresolved theoretical matters are left for future work.

6.1 Acknowledgments

Special thanks to Agnieska Międ̨lar for her collaboration on nonlinear eigenvalue problems. The work in this dissertation was supported by the National Science Foundation under grant CCF #1510010, and by Intel Corporation.

APPENDIX A

SUBSPACE ITERATIONS CONVERGENCE

This appendix shows how to derive the upper bound on the eigenvector error for Inexact FEAST

$$||x_j - \tilde{q}_j|| \leq \left(\frac{|\gamma_{m_0+1}| + \alpha_j \Delta}{|\gamma_j|} \right) \; ||x_j - q_j||, \tag{A.1}$$

which shows that we can expect the FEAST algorithm to converge linearly even when its associated linear systems of equations are solved inexactly.

The first section reviews the analysis of standard Subspace Iteration and its application to the FEAST algorithm. Section A.2 then shows how this analysis can be adjusted to accommodate for errors in matrix multiplication, which allows us to account for the inexact solution of the FEAST linear systems of equations. Section A.2 discusses the analysis of IFEAST for standard eigenvalue problems, and Section A.3 adjusts this analysis for generalized eigenvalue problems.

A.1 Standard Subspace Iterations

I will first recount the treatment of the convergence of standard subspace iterations in Section 2 of [74], before showing how this treatment can be modified to describe the convergence behavior of FEAST and IFEAST.

With standard subspace iterations, we want to find the eigenvectors corresponding to the m_0 largest-magnitude eigenvalues of a matrix $A \in \mathbb{C}^{n \times n}$. This is done by repeatedly multiplying an approximate eigenvector subspace basis $Q \in \mathbb{C}^{n \times m_0}$ by

A, and reorthogonalizing the column vectors of Q in between multiplications (using Rayleigh-Ritz, for example). The usual method for proving convergence is to show that, for every desired eigenvector $x_j,\ 1 \leq j \leq m_0$, an upper bound on the error of its estimation in the subspace $\mathcal{R}(Q)$ goes down after each subspace iteration, where I denote the range of the column vectors of Q by $\mathcal{R}(Q)$. This can be done by judiciously choosing a vector $q_j \in \mathcal{R}(Q)$ that is close to x_j and showing that there is always a different vector $\tilde{q}_j \in \mathcal{R}(AQ)$ that is closer to x_j than q_j is.

Let $X_1 \in \mathbb{C}^{n \times m_0}$ be the matrix whose column vectors are the eigenvectors that we want to find, and $X_2 \in \mathbb{C}^{n \times (n-m_0)}$ be the matrix composed of the other $n - m_0$ eigenvectors. Then the vector q_j is usually chosen to be the vector in $\mathcal{R}(Q)$

$$q_j = w_j + x_j, \tag{A.2}$$

where $w_j \in \mathcal{R}(X_2)$. Such a vector $q_j \in \mathcal{R}(Q)$ is guaranteed to exist and be unique [74] provided that $rank(PQ) = rank(X_1)$, where P is any projector on to $\mathcal{R}(X_1)$. The vector $\tilde{q}_j$ is then chosen to be

$$\tilde{q}_j = \frac{1}{\lambda_j} A q_j, \tag{A.3}$$

where $\lambda_j \neq 0$ is the eigenvalue corresponding to the eigenvector x_j. The difference vector $\tilde{w}_j = \tilde{q}_j - x_j$ is then also an element of the subspace spanned by X_2, a fact that we can use to relate $||w_j||$ to $||\tilde{w}_j||$ i.e.

$$\tilde{q}_j = \frac{1}{\lambda_j} A q_j = \frac{1}{\lambda_j}(A x_j + A w_j) = x_j + \frac{1}{\lambda_j} A w_j, \tag{A.4}$$

$$\tilde{w}_j = \tilde{q}_j - x_j = \frac{1}{\lambda_j} A w_j, \tag{A.5}$$

$$||\tilde{w}_j|| = \frac{1}{|\lambda_j|} ||A w_j|| \leq \frac{|\lambda_{m_0+1}|}{|\lambda_j|} ||w_j||, \tag{A.6}$$

where we know that $||Aw_j|| \leq |\lambda_{m_0+1}|||w_j||$ because w_j is in the subspace spanned by X_2, the $(n - m_0)$ eigenvectors corresponding to the eigenvalues with magnitudes less than or equal to $|\lambda_{m_0+1}|$. Equation (A.6) shows that an upper bound on the error for the estimation of x_j in the subspace spanned by Q always decreases when Q is multiplied by A, and that it does so at a rate that is linear and proportional to the ratio between $|\lambda_{m_0+1}|$ and $|\lambda_j|$. Thus, subspace iterations are guaranteed to converge faster when the subspace basis Q has a larger dimension and when the eigenvalues of A are more separated.

A.2 Inexact FEAST

We can find a similar upper bound with which to analyze the convergence of IFEAST by following a similar line of reasoning. Standard FEAST can be interpreted as a subspace iteration that uses the matrix $\rho(A)$ instead of the original matrix A,

$$A \longrightarrow \rho(A) = \sum_{k=1}^{n_c} \omega_k (z_k I - A)^{-1}. \tag{A.7}$$

Then the upper bound (A.6) becomes

$$||\tilde{w}_j|| \leq \frac{|\gamma_{m_0+1}|}{|\gamma_j|} ||w_j||, \tag{A.8}$$

where γ_j is the j^{th} largest eigenvalue of $\rho(A)$, with corresponding eigenvector x_j. The γ_j with the largest magnitudes correspond to the 'wanted' eigenvalues of A that lie inside of the FEAST integration contour. Making the FEAST quadrature rule more accurate by increasing the number of quadrature points n_c has the effect of making the ratio $|\gamma_{m_0+1}|/|\gamma_j|$ smaller, which is how standard FEAST can improve its rate of convergence by solving more linear systems.

Equation (A.8) requires modification when the linear systems of FEAST are solved inexactly. In particular, if we apply $\rho(A)$ by solving the linear systems

$$\frac{1}{\omega_k}(z_k I - A)y_{k,j} = q_j,\ \forall k = 1, \ldots, n_c,\ \forall j = 1, \ldots, m_0 \tag{A.9}$$

such that there is some error $s_{k,j}$ in the solution of the linear system, i.e.

$$s_{k,j} = \omega_k(z_k I - A)^{-1} q_j - y_{k,j}, \tag{A.10}$$

then, for IFEAST, equation (A.3) becomes

$$\tilde{q}_j = \frac{1}{\gamma_j}\left(\rho(A)q_j - \sum_{k=1}^{n_c} s_{k,j}\right). \tag{A.11}$$

This is not necessarily very useful in practice, however, because the values of $s_{k,j}$ are not known. Instead, (A.11) can be rewritten in terms of the linear system residuals, the norms of which are used as the stopping criteria for iterative linear system solvers:

$$\tilde{q}_j = \frac{1}{\gamma_j}\left(\rho(A)q_j - \sum_{k=1}^{n_c} \omega_k(z_k I - A)^{-1} r_{k,j}\right), \tag{A.12}$$

with

$$r_{k,j} = q_j - \frac{1}{\omega_k}(z_k I - A)y_{k,j}. \tag{A.13}$$

Since $q_j = w_j + x_j$, we can derive the expression for $\tilde{w}_j$ from (A.12):

$$\tilde{w}_j = \tilde{q}_j - x_j = \frac{1}{\gamma_j}\left(\rho(A)w_j - \sum_{k=1}^{n_c} \omega_k(z_k I - A)^{-1} r_{k,j}\right). \tag{A.14}$$

We can then find an upper bound similar to (A.8):

$$||\tilde{w}_j|| \leq \frac{|\gamma_{m_0+1}|}{|\gamma_j|}||w_j|| + \frac{1}{|\gamma_j|}\sum_{k=1}^{n_c} ||\omega_k(z_k I - A)^{-1}||\ ||r_{k,j}||. \tag{A.15}$$

Assuming that all the linear systems (A.9) are solved using iterative solvers with the same tolerance ϵ on the residual norm, then $||r_{k,j}|| \leq \epsilon,\ \forall k, j$, we get:

$$||\tilde{w}_j|| \leq \left(\frac{|\gamma_{m_0+1}| + \alpha_j \Delta}{|\gamma_j|} \right) ||w_j||, \tag{A.16}$$

with

$$\alpha_j = \epsilon / ||w_j||, \tag{A.17}$$

and

$$\Delta = \sum_{k=1}^{n_c} ||\omega_k (z_k I - A)^{-1}||. \tag{A.18}$$

If the linear systems are solved such that α_j is the same at every FEAST subspace iteration, then linear convergence is guaranteed, with the rate of convergence depending on accuracy of the linear system solutions.

A.3 Inexact FEAST and Generalized Eigenvalue Problems

The eigenvector error bound in Equation (A.15) can be extended to the case of Inexact FEAST for generalized eigenvalue problems $Ax = \lambda Bx$ as well. The entire analysis of the preceding section applies directly to the standardization $B^{-1}Ax = \lambda x$; the only difference is that the value of Δ changes due to the way that the linear systems of equations are solved. IFEAST for the generalized eigenvalue problem approximately solves the linear systems of equations

$$\frac{1}{\omega_k}(z_k B - A) y_{k,j} = B q_j, \tag{A.19}$$

rather than using the standardization $\frac{1}{\omega_k}(z_k - B^{-1}A) y_{k,j} = q_j$ (i.e. using Equation (A.9) directly). The residuals for the approximate solution to Equation (A.19) are

$$r_{k,j}^{(G)} = B q_j - \frac{1}{\omega_k}(z_k B - A) y_{k,j}, \tag{A.20}$$

which are notably different from the residuals $r_{k,l}$ in Equation (A.13). For the same approximate solution $y_{k,l}$, $r_{k,j} = B^{-1}r_{k,l}^{(G)}$. The linear system errors $s_{k,j}$ are then

$$s_{k,j} = \omega_k(z_k I - B^{-1}A)^{-1} r_{k,j} \tag{A.21}$$

$$= \omega_k(z_k B - A)^{-1} r_{k,j}^{(G)}, \tag{A.22}$$

and Equation (A.14) becomes

$$\tilde{w}_j = \tilde{q}_j - x_j = \frac{1}{\gamma_j}\left(\rho(A)w_j - \sum_{k=1}^{n_c} \omega_k(z_k B - A)^{-1} r_{k,j}^{(G)}\right). \tag{A.23}$$

The rest of the analysis proceeds in exactly the same way, with the result

$$||\tilde{w}_j|| \leq \left(\frac{|\gamma_{m_0+1}| + \alpha_j \Delta}{|\gamma_j|}\right) ||w_j||, \tag{A.24}$$

$$\alpha_j = \epsilon / ||w_j||, \tag{A.25}$$

$$\Delta = \sum_{k=1}^{n_c} ||\omega_k(z_k B - A)^{-1}||, \tag{A.26}$$

where now ϵ is the tolerance on the norm $||r_{k,l}^{(G)}||$. In the rest of this dissertation I generally refer to the linear system residuals without the qualifying (G) superscript, since it is understood that the IFEAST linear systems of equations always take the form of Equation (A.19) rather than $\frac{1}{\omega_k}(z_k - B^{-1}A)y_{k,j} = q_j$.

APPENDIX B

THE CONTOUR INTEGRAL FOR GENERALIZED IFEAST

The contour integral for the standard FEAST algorithm turns out to be inefficient for solving generalized eigenvalue problems or preconditioned eigenvalue problems with the IFEAST algorithm, in which the FEAST linear systems of equations are deliberately solved inexactly. The number of linear system iterations that is required to solve the IFEAST linear systems increases with each IFEAST iteration as the eigenvector solutions converge, resulting in a substantial total computational cost for solving the eigenvalue problem.

An alternative variation of the FEAST algorithm, which I refer to as Generalized IFEAST, uses a modified contour integral so that the number of required linear system iterations at each IFEAST iteration is constant, even as the eigenvector solutions converge. This appendix describes how the Generalized IFEAST contour integration is derived, and shows that it allows the Basic IFEAST convergence heuristic to be satisfied when solving linear systems of equations to a constant level of accuracy, even for generalized eigenvalue problems or preconditioned linear systems.

Section B.1 derives the Generalized IFEAST contour integral from the standard FEAST contour integral, showing that the two are equal when their linear systems of equations are solved exactly. Section B.2 shows how this contour integral is related to restarting the solution of linear systems of equations. Section B.3 uses the relationship between the Generalized IFEAST contour integral and linear system restarts to show that solving the Generalized IFEAST linear systems of equations to a constant level

of accuracy is equivalent to solving the Basic IFEAST linear systems of equations to an increasing level of accuracy. This ensures that the convergence heuristic for Basic IFEAST can always be satisfied efficiently, even when solving generalized eigenvalue problems or using preconditioners.

B.1 Deriving the Generalized IFEAST Integral

For a linear, generalized eigenvalue problem

$$Ax = \lambda Bx, \tag{B.1}$$

the standard FEAST algorithm calculates the eigenvectors whose eigenvalues lie inside a closed contour $\mathcal{C}$ in the complex plane by using contour integration to form a subspace that approximately spans those eigenvectors, i.e.

$$Q = \frac{1}{2\pi i} \oint_{\mathcal{C}} (zB - A)^{-1} B \tilde{X}^{(0)} dz, \tag{B.2}$$

where $\tilde{X}^{(0)} \in \mathbb{C}^{n \times m_0}$ is a (possibly random) initial guess for the eigenvectors of interest.

The Generalized IFEAST algorithm, on the other hand, uses the integral

$$\frac{1}{2\pi i} \oint_{\mathcal{C}} (X^{(0)} - (zB - A)^{-1} R_E)(zI - \Lambda)^{-1} dz, \tag{B.3}$$

where $R_E = B\tilde{X}^{(0)}\tilde{\Lambda} - A\tilde{X}^{(0)}$ is the block eigenvector residual for the initial guess $\tilde{X}^{(0)}$, and $\tilde{\Lambda}$ is the diagonal matrix of Ritz values for $\tilde{X}^{(0)}$.

We can show that Equations (B.2) and (B.3) are equal by using a simple trick, first introduced in [31] for use in the case of single vector Shift and Invert Itera-

tion. The integral (B.2) is an integration of shifted linear system solutions that are parameterized by their location in the complex plane, i.e.

$$Q = \frac{1}{2\pi i} \oint_{\mathcal{C}} Q(z) dz \tag{B.4}$$

such that

$$(zB - A)Q(z) = B\tilde{X}^{(0)}. \tag{B.5}$$

The trick is to assert that the parameterized solution $Q(z)$ must take the form

$$Q(z) = (\tilde{X}^{(0)} + \delta Q(z))(zI - \tilde{\Lambda})^{-1}, \tag{B.6}$$

where $\tilde{X}^{(0)}$ are Ritz vectors for the eigenvalue problem (B.1) (generated from, for example, a set of m_0 random initial guess vectors), $\tilde{\Lambda}$ is the diagonal matrix of the corresponding Ritz values, and $\delta(z) \in \mathbb{C}^{n \times m_0}$ is unknown. Thus, rather than finding $Q(z)$ by solving the linear systems (B.5), we can instead find $\delta Q(z)$ by solving the linear system

$$(zB - A)(\tilde{X}^{(0)} + \delta Q(z))(zI - \tilde{\Lambda})^{-1} = B\tilde{X}^{(0)}, \tag{B.7}$$

which is just equation (B.6) inserted into equation (B.5). We can then use $\delta Q(z)$ to calculate $Q(z)$ using equation (B.6).

Equation (B.7) can be simplified algebraically as follows:

$$(zB - A)(X^{(0)} + \delta Q(z))(zI - \Lambda)^{-1} = BX^{(0)} \tag{B.8}$$

$$(zB - A)(X^{(0)} + \delta Q(z)) = BX^{(0)}(zI - \Lambda) \tag{B.9}$$

$$(zB - A)\delta Q(z) = zBX^{(0)} - BX^{(0)}\Lambda - zBX^{(0)} + AX^{(0)} \tag{B.10}$$

$$(zB - A)\delta Q(z) = -(BX^{(0)}\Lambda - AX^{(0)}) = -R_E \tag{B.11}$$

Thus $\delta Q(z) = -(zB - A)^{-1}R_E$, the right hand side of which converges to zero as $\tilde{X}$ converges to a block of eigenvectors. $Q(z)$ then evaluates to

$$Q(z) = (\tilde{X}^{(0)} - (zB - A)^{-1}R_E)(zI - \Lambda)^{-1} \tag{B.12}$$

Inserting this expression into equation (B.4) gives equation (B.3).

B.2 Connection with Linear System Restarts

Linear system restarts (see Section 2.2.5) are a method for refining an approximate solution to a linear system of equations. The integral in Equation (B.3) can be seen as the result of restarting the solution of the FEAST linear systems of equations by using the estimates that FEAST provides for eigenvalues and eigenvectors.

Equation (B.6) is equivalent to using iterative refinement for the solution of Equation (B.5) with the initial guess

$$\tilde{Q}_{old}(z) = \tilde{X}^{(0)}(zI - \tilde{\Lambda})^{-1}. \tag{B.13}$$

Because we are solving an eigenvalue problem, this is the most sensible initial guess that we could come up with based on the available information; in the case that $\tilde{X}^{(0)}$ and $\tilde{\Lambda}$ are the exact eigenvalues and eigenvectors, $\tilde{Q}(z)$ in Equation (B.13) is the exact solution to the linear system in Equation (B.5).

Traditional restarts would calculate a new approximate solution as $\tilde{Q}_{new}(z) = \tilde{Q}(z) + \delta Q$. Equation (B.6), instead, is a modified restarting procedure in which the update δQ is asserted to have the same form as the original estimated solution, i.e.

$$\tilde{Q}_{new}(z) = \tilde{Q}_{old}(z) + \delta Q(zI - \Lambda)^{-1}. \tag{B.14}$$

The linear system residuals of $\tilde{Q}(z)$ as an approximate solution for Equation (B.5) are equal to the eigenvector residuals of $\tilde{X}^{(0)}$ and $\tilde{\Lambda}$ as an approximate solution to $AX = BX\Lambda$, up to multiplication by a constant:

$$R_L = B\tilde{X}^{(0)} - (zB - A)\tilde{Q}_{old}(z) \tag{B.15}$$

$$= B\tilde{X}^{(0)} - (zB - A)\tilde{X}^{(0)}(zI - \tilde{\Lambda})^{-1} \tag{B.16}$$

$$= \left(B\tilde{X}^{(0)}(zI - \tilde{\Lambda}) - (zB - A)\tilde{X}^{(0)}\right)(zI - \tilde{\Lambda})^{-1} \tag{B.17}$$

$$= \left(A\tilde{X}^{(0)} - B\tilde{X}^{(0)}\tilde{\Lambda}\right)(zI - \tilde{\Lambda})^{-1} \tag{B.18}$$

$$= -R_E(zI - \tilde{\Lambda})^{-1}, \tag{B.19}$$

where I use R_L and R_E to distinguish between the linear system residuals and the eigenvector residuals, respectively.

The linear system residuals for the iteratively-refined solution, $\tilde{Q}_{new}(z)$ in Equation (B.14), are the residuals for the solution of the correction equation

$$(zB - A)\delta Q(zI - \Lambda)^{-1} = BX^{(0)} - (zB - A)\tilde{Q}_{old}(z). \tag{B.20}$$

If Equation (B.20) is solved approximately for an approximate solution $\delta\tilde{Q}$ such that its linear system residual is

$$B\tilde{X}^{(0)} - (zB - A)\tilde{Q}_{old}(z) - (zB - A)\delta\tilde{Q}(zI - \Lambda)^{-1} = \delta R \tag{B.21}$$

then the final residual for the original FEAST linear system in Equation (B.5), using the approximate solution in Equation (B.14), is δR, just as with typical linear system restarts.

B.3 Generalized IFEAST Linear System Accuracy

The motivation for using the integral

$$\frac{1}{2\pi i}\oint_{\mathcal{C}}(\tilde{X}^{(0)}-(zB-A)^{-1}R_E)(zI-\tilde{\Lambda})^{-1}dz \tag{B.22}$$

is to be able to get away with using a relatively small, constant number matrix multiplications when approximating the FEAST linear system solutions at each FEAST iteration. We can see why this works by using the fact that the linear system residuals of FEAST are closely related to the eigenvector residual when using the integral in Equation (B.22).

As discussed in the previous section, Equation (B.22) is equivalent to solving the FEAST linear systems of equations

$$(zB-A)Q(z)=B\tilde{X}^{(0)} \tag{B.23}$$

by using an initial guess

$$\tilde{Q}_{old}(z)=\tilde{X}^{(0)}(zI-\tilde{\Lambda})^{-1} \tag{B.24}$$

in conjunction with a specific form of update equation for restarting, i.e.

$$\tilde{Q}_{new}(z)=\tilde{Q}_{old}(z)+\delta Q(z)(zI-\tilde{\Lambda})^{-1}. \tag{B.25}$$

Rather than solving Equation (B.23) directly for $Q(z)$, one instead solves the correction equation

$$(zB-A)\delta Q(z)(zI-\tilde{\Lambda})^{-1}=B\tilde{X}^{(0)}-(zB-A)\tilde{Q}_{old}(z). \tag{B.26}$$

for the correction $\delta Q(z)$, and finds a new approximation for $Q(z)$ using Equation (B.25).

In practice Equation (B.26) is solved approximately such that

$$(zB - A)\delta\tilde{Q}(z)(zI - \tilde{\Lambda})^{-1} = BX^{(0)} - (zB - A)\tilde{Q}_{old}(z) - \delta R, \tag{B.27}$$

where $\delta\tilde{Q}(z)$ is an approximate solution and δR is the corresponding linear system residual. Because this is just a particular form of restart, the final residual for the original linear system of equations (B.23) with approximate solution $\tilde{Q}_{new}(z)$ from Equation (B.25) is also δR.

If δr_i is the i^{th} column vector of δR and $r_i^{(L)}$ is the i^{th} column vector of $B\tilde{X}^{(0)} - (zB - A)\tilde{Q}_{old}(z)$, i.e. it is a linear system residual for the initial guess $\tilde{Q}_{old}(z)$, then

$$\frac{||\delta r_i||}{||r_i^{(L)}||} < \epsilon_i \tag{B.28}$$

is the convergence condition for the approximate solution of the correction equation (B.26), with ϵ being the user-determined tolerance on the relative residual. The quantity $||\delta r_i||/||r_i^{(L)}||$ is called the "relative residual", and it is the quantity that iterative linear system solvers naturally use to determine whether or not a prospective solution is sufficiently accurate; dividing by the norm of the right hand side $||r_i^{(L)}||$ removes the right hand side magnitude from the consideration about whether or not the linear system solution has converged, which is important because the norm of the right hand side plays no role in the solution of linear systems of equations for most iterative solving methods.

As noted in the previous section, the linear system residuals are the same as the eigenvector residuals (up to a scalar multiple) for the initial guess $\tilde{Q}_{old}(z)$ in Equation (B.24). The linear system stopping condition in Equation (B.28) is thus equivalent to

$$||\delta r_i|| < \frac{\epsilon}{|z - \tilde{\lambda}_i|}||r_i^{(E)}||, \tag{B.29}$$

where $r_i^{(E)} = \tilde{\lambda}_i B\tilde{x}_i^{(0)} - A\tilde{x}_i^{(0)}$ is the i^{th} eigenvector residual.

Using Equation (B.29) allows us to draw a connection between the requirements of the IFEAST algorithm and the amount of work that is necessary to solve the correction equation (B.26), and therefore to evaluate the integral (B.22). We expect that IFEAST will converge linearly provided that the linear systems of equations (B.23) are solved such that their residual norms are some fraction α of the current eigenvector residual norms, i.e.

$$||\delta r_i|| < \alpha ||r_i^{(E)}||, \tag{B.30}$$

where we have used the fact that we know the linear system residuals will be the residuals δr_i from solving the correction equation (B.26) approximately. Setting the right hand side of Equation (B.29) to be less than the right hand side of Equation (B.30) gets us the condition on the correction equation tolerance ϵ_i that will ensure that the inequality in Equation (B.30) is always satisfied:

$$\epsilon_i < \alpha |z - \tilde{\lambda}_i|. \tag{B.31}$$

The implication of Equation (B.31) is that, once IFEAST iterations have converged enough that the distances between the approximate eigenvalues $\tilde{\lambda}_i$ and the linear system shifts z do not change much, the maximum values of the linear system tolerances ϵ_i that will guarantee convergence of the eigenvalue problem become constant. Thus, when using the contour integral in Equation (B.22), rather than the traditional FEAST contour, the linear systems of equations can be solved to a *constant* level of accuracy in order to ensure the convergence of an eigenvalue solution. In many cases the viable values of ϵ_i can be quite high. It is not uncommon for $\epsilon_i = 10^{-2}$, for example, meaning that only a relatively small amount of work needs to be done to solve the linear systems of equations at each IFEAST iteration.

APPENDIX C

GMRES ITERATIONS AND EIGENVECTOR RESIDUALS

One of the appealing features of both Inexact Shift and Invert Iteration and Basic IFEAST is that, for the standard eigenvalue problem $Ax = \lambda x$, each of these algorithms is able to converge linearly to a solution by using a constant number of linear system iterations at each outer eigenvalue iteration. This appendix describes one approach for understanding why this happens. I focus primarily on the GMRES algorithm, simply because it allows for easy analysis. That GMRES with IFEAST requires a constant number of iterations is a consequence of there being an upper bound on the number of GMRES iterations that is required to adequately refine approximate eigenvector solutions, and this upper bound being independent of how close to convergence those approximate eigenvectors are.

C.1 Upper Bound on Linear System Iterations

For the standard eigenvalue problem $Ax = \lambda x$, both Inexact Shift and Invert Iteration and Basic IFEAST require the solution of linear systems of equations that take the form

$$(zI - A)y = \tilde{x}_s, \tag{C.1}$$

where A is an $n \times n$ matrix, z is a shift in the complex plane, and $\tilde{x}_s$ is an estimation for the exact eigenvector x_s with corresponding eigenvalue λ_s. The convergence theory for both Inexact Shift and Invert Iteration and Basic IFEAST (see Appendix A) guarantees linear convergence of the eigenvalue problem provided that the linear

systems of equations in either algorithm are solved such that their residual tolerances are a constant fraction of the eigenvector error at a given iteration, i.e.

$$||\tilde{x}_s - (zI - A)\tilde{y}|| \leq \alpha||\tilde{x}_s - x_s||, \tag{C.2}$$

where $\tilde{y}$ is an approximate solution to Equation (C.1), and α is a real number between 0 and 1.

If k iterations of any Krylov subspace algorithm are used to approximate Equation (C.1), then $\tilde{y}$ takes the form

$$\tilde{y} = p_k(zI - A)\tilde{x}_s \tag{C.3}$$

with $p_k(\lambda)$ being a k-degree polynomial, and the linear system residual r_L is

$$r_L = \tilde{x}_s - (zI - A)p_k(zI - A)\tilde{x}_s = q_k(zI - A)x_s, \tag{C.4}$$

where $q_k(\lambda) = 1 - \lambda p_k(\lambda)$ is a k-degree polynomial that is constrained such that $q_k(0) = 1$. The polynomial $q_k(\lambda)$ can be written in terms of its zeros w_i as

$$q_k(\lambda) = \prod_{i=1}^{k} \frac{1}{w_i}(\lambda - w_i), \tag{C.5}$$

and the approximate eigenvector $\tilde{x}_s$ can be written in terms of the exact eigenvectors x_i as

$$\tilde{x}_s = \sum_{i=1}^{n} c_i x_i. \tag{C.6}$$

We can assume that $\tilde{x}_s$ is normalized such that $c_s = 1$. Combining Equations (C.4), (C.5), and (C.6) we get

$$r_L = q_k(zI - A)x_s = \sum_j \prod_{i=1}^{k} \frac{1}{w_i}([z - \lambda_j] - w_i)c_j x_j \tag{C.7}$$

We can get an upper bound on $||r_L||^2$ by choosing a particular form for $q_k(\lambda)$. If $\tilde{x}_s$ is already converging towards x_s, then $|c_s| > |c_i|$ for all $i \neq s$, and we can choose

the k^{th} zero of $q_k(\lambda)$ to be $w_k = z - \lambda_s$ in order to eliminate the component of r_L that is in the direction of x_s. The linear system residual becomes

$$r_L = \sum_{j \neq s} \frac{\lambda_s - \lambda_j}{z - \lambda_s} \prod_{i=1}^{k-1} \frac{1}{w_i} ([z - \lambda_j] - w_i) c_j x_j \tag{C.8}$$

$$= \sum_{j \neq s} \frac{\lambda_s - \lambda_j}{z - \lambda_s} q_{k-1}(z - \lambda_j) c_j x_j, \tag{C.9}$$

where $q_{k-1}(\lambda)$ is a $(k-1)$-degree polynomial with $q_{k-1}(0) = 1$.

With both Inexact Shift and Invert Iteration and Basic IFEAST, the solution $\tilde{x}_s$ will converge linearly to x_s provided that

$$||r_L||^2 < \alpha^2 ||\tilde{x}_s - x_s||^2 \tag{C.10}$$

is true for a sufficiently low value of α (see Section A.2 on page 174 for more details about the relationship between α and IFEAST convergence). Using Equations (C.9) and (C.6), this condition becomes

$$\sum_{i,j \neq s} \frac{(\lambda_s - \lambda_i)^* (\lambda_s - \lambda_j)}{|z - \lambda_s|^2} q_{k-1}(z - \lambda_i)^* q_{k-1}(z - \lambda_j) c_i^* c_j x_i^H x_j < \sum_{i,j \neq s} \alpha^2 c_i^* c_j x_i^H x_j. \tag{C.11}$$

A sufficient condition for Equation (C.11) to be true is

$$\left| \frac{(\lambda_s - \lambda_i)^* (\lambda_s - \lambda_j)}{|z - \lambda_s|^2} q_{k-1}(z - \lambda_i)^* q_{k-1}(z - \lambda_j) \right| < \alpha^2 \quad \forall \;\; i, j. \tag{C.12}$$

This condition is notably independent of the eigenvector error $||\tilde{x}_s - x_s||$; it depends only on the eigenvalues of A (and indirectly on the eigenvectors through α, for non-symmetric problems). The largest value of k that makes it true is therefore a constant over all eigenvector iterations, which implies the observed behavior wherein the number of required linear system iterations is roughly constant when using Inexact Shift and Invert Iteration or Basic IFEAST.

APPENDIX D

TWO-SIDED FEAST

This Appendix provides an outline of the two-sided Generalized IFEAST Algorithm, which is used for solving the Grcar problem in Section 4.6.5 on page 138. The two-sided FEAST algorithm follows essentially the same steps as the one-sided FEAST algorithm, but it solves two eigenvalue problems simultaneously: one for the right eigenvectors, and one for the left eigenvectors. The most significant difference between one-sided FEAST and two-sided FEAST is that two-sided FEAST B-biorthogonalizes the approximate left and right eigenvector subspaces at each iteration, which substantially improves the robustness of the algorithm for nonsymmetric problems. The two-sided Generalized IFEAST algorithm is given in Algorithm 11.

Algorithm 11 The Two-sided Generalized IFEAST algorithm for solving $AX = BX\Lambda$

Inputs:

- Matrices $A, B \in \mathbb{C}^{n\times n}$
- Closed contour $\mathcal{C}$ that encloses the search region for eigenvalues in the complex plane
- Overestimate m_0 for the number of eigenvalues inside $\mathcal{C}$
- Initial guesses $\tilde{X}_R^{(0)}$ and $\tilde{X}_L^{(0)} \in \mathbb{C}^{n\times m_0}$ for the right and left eigenvector subspaces
- Set of n_c quadrature weights and points (ω_k, z_k) for numerically integrating equation (4.1)
- Tolerance α for linear system relative residuals, with $0 < \alpha < 1$

0. Set $Q_R = \tilde{X}_R^{(0)}$, $Q_L = \tilde{X}_L^{(0)}$

For each subspace iteration i:

1. B-Biorthogonalize Q_R and Q_L using e.g. the SVD:

$$U\Sigma V^H = Q_L^H B Q_R \;\longrightarrow\; Q_R = Q_R V \Sigma^{-\frac{1}{2}}, \;\; Q_L = Q_L U \Sigma^{-\frac{1}{2}} \tag{D.1}$$

2. Perform Rayleigh-Ritz procedure to find a new estimate for eigenvalues and eigenvectors:

i. Solve reduced eigenvalue problem for $X_Q \in \mathbb{C}^{m_0\times m_0}$

$$A_Q X_{QR} = B_Q X_{QR}\tilde{\Lambda}, \quad A_Q^H X_{QL} = B_Q^H X_{QL}\tilde{\Lambda}^*$$

with $A_Q = Q_L^H A Q_R$ and $B_Q = Q_L^H B Q_R$

ii. Get new estimates for subspaces: $\tilde{X}_R^{(i)} = Q_R X_{QR}$, $\tilde{X}_L^{(i)} = Q_L X_{QL}$

3. Calculate the FEAST eigenvector residuals $R_R = B\tilde{X}_R^{(i)}\tilde{\Lambda} - A\tilde{X}_R^{(i)}$, $R_L = B^H\tilde{X}_L^{(i)}\tilde{\Lambda}^* - A^H\tilde{X}_L^{(i)}$. If

$$\max_{1\le j\le m_0} \; ||A\tilde{x}_{Rj}^{(i)} - \tilde{\lambda}_j B\tilde{x}_{Rj}^{(i)}||, \;\; \tilde{\lambda}_j \text{ inside } \mathcal{C} \tag{D.2}$$

is below a given tolerance, **EXIT**.

4. Iteratively solve n_c shifted linear systems for two contour integrals, each with m_0 right hand sides, for Y_{Rk} and Y_{Lk} .

$$\frac{1}{\omega_k}(z_k B - A)Y_{Rk}^{(i)} = R_R, \quad \frac{1}{\omega_k}(z_k B - A)^H Y_{Lk}^{(i)} = R_L, \;\; 1 \le k \le n_c \tag{D.3}$$

such that the iterations are stopped when the following tolerance on the linear system residuals is met:

$$||R_R e_j - \frac{1}{\omega_k}(z_k B - A)Y_{Rk}^{(i)} e_j|| \le \alpha \;\; \forall j,\; 1 \le j \le m_0,$$

with the corresponding tolerance being used for the left eigenspace solutions $Y_{Rk}^{(i)}$.

5. Form the filtered subspaces Q_R and Q_L

$$Q_R = \sum_{k=1}^{n_c} \omega_k \left(\tilde{X}_R^{(i)} - Y_{Rk}^{(i)}\right)(z_k I - \tilde{\Lambda})^{-1}, \quad Q_L = \sum_{k=1}^{n_c} \omega_k^* \left(\tilde{X}_L^{(i)} - Y_{Lk}^{(i)}\right)(z_k^* I - \tilde{\Lambda}^*)^{-1},$$

6. **GOTO** Step 1.

Outputs: diagonal matrix $\tilde{\Lambda}$ of approximations for the m eigenvalues inside $\mathcal{C}$, and approximations for the corresponding right and left eigenvectors $\tilde{X}_R$ and $\tilde{X}_L$.

BIBLIOGRAPHY

[1] Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., and Sorensen, D. *LAPACK Users' Guide*, third ed. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1999.

[2] Asakura, J., Sakurai, T., Tadano, H., Ikegami, T., and Kimura, K. A numerical method for nonlinear eigenvalue problems using contour integrals. *JSIAM Lett. 1* (2009), 52–55.

[3] Bai, Zhaojun, and Su, Yangfeng. SOAR: A second-order Arnoldi method for the solution of the quadratic eigenvalue problem. *SIAM J. Matrix Anal. Appl. 26*, 3 (2005), 640–659.

[4] Baker, Chris G, Hetmaniuk, Ulrich L, Lehoucq, Richard B, and Thornquist, Heidi K. Anasazi software for the numerical solution of large-scale eigenvalue problems. *ACM Transactions on Mathematical Software (TOMS) 36*, 3 (2009), 13.

[5] Belkin, Mikhail, and Niyogi, Partha. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in neural information processing systems* (2002), pp. 585–591.

[6] Berns-Müller, Jörg, Graham, Ivan G, and Spence, Alastair. Inexact inverse iteration for symmetric matrices. *Linear Algebra and its Applications 416*, 2-3 (2006), 389–413.

[7] Betcke, T., Higham, N. J., Mehrmann, V., Schröder, C., and Tisseur, F. NLEVP: a collection of nonlinear eigenvalue problems. *ACM Trans. Math. Software 39*, 2 (2013), Art. 7, 28.

[8] Betcke, Timo, and Voss, Heinrich. A Jacobi–Davidson-type projection method for nonlinear eigenvalue problems. *Future Generation Computer Systems 20*, 3 (2004), 363–372.

[9] Beyn, W. J. An integral method for solving nonlinear eigenvalue problems. *Linear Algebra Appl. 436*, 10 (2012), 3839–3863.

[10] Beyn, W. J., Effenberger, C., and Kressner, D. Continuation of eigenvalues and invariant pairs for parameterized nonlinear eigenvalue problems. *Numer. Math. 119*, 3 (2011), 489–516.

[11] Bollhöfer, Matthias, and Notay, Yvan. Jadamilu: a software code for computing selected eigenvalues of large sparse symmetric matrices. *Computer Physics Communications 177*, 12 (2007), 951–964.

[12] Bridges, T. J., and Morris, P.J. Differential eigenvalue problems in which the parameter appears nonlinearly. *J. Comput. Phys. 55*, 3 (1984), 437–460.

[13] Choi, Jaeyoung, Dongarra, Jack J, Pozo, Roldan, and Walker, David W. Scalapack: A scalable linear algebra library for distributed memory concurrent computers. In *Frontiers of Massively Parallel Computation, 1992., Fourth Symposium on the* (1992), IEEE, pp. 120–127.

[14] Christopher, M Bishop. *Pattern Recognition and Machine Learning.* Springer-Verlag New York, 2016.

[15] Davis, T.A., and Hu, Y. The university of florida sparse matrix collection. *ACM Transactions on Mathematical Software 38*, 1 (2011).

[16] Fletcher, Roger. Conjugate gradient methods for indefinite systems. In *Numerical analysis*. Springer, 1976, pp. 73–89.

[17] Freitag, MA, and Spence, A. Convergence of inexact inverse iteration with application to preconditioned iterative solves. *BIT Numerical Mathematics 47*, 1 (2007), 27–44.

[18] Freitag, MA, and Spence, Alastair. Rayleigh quotient iteration and simplified jacobi–davidson method with preconditioned iterative solves. *Linear Algebra and its Applications 428*, 8-9 (2008), 2049–2060.

[19] Freitag, Melina A, Kürschner, Patrick, and Pestana, Jennifer. Gmres convergence bounds for eigenvalue problems. *Computational Methods in Applied Mathematics 18*, 2 (2018), 203–222.

[20] Freitag, Melina A, and Spence, Alastair. A tuned preconditioner for inexact inverse iteration applied to hermitian eigenvalue problems. *IMA journal of numerical analysis 28*, 3 (2008), 522–551.

[21] Freund, Roland. On conjugate gradient type methods and polynomial preconditioners for a class of complex non-hermitian matrices. *Numerische Mathematik 57*, 1 (1990), 285–312.

[22] Galgon, Martin, Kraemer, Lukas, Thies, Jonas, Basermann, Achim, and Lang, Bruno. On the parallel iterative solution of linear systems arising in the feast algorithm for computing inner eigenvalues. *Parallel Computing 49* (2015), 153–163.

[23] Gavin, Brendan, Międlar, Agnieszka, and Polizzi, Eric. Feast eigensolver for nonlinear eigenvalue problems. *Journal of Computational Science 27* (2018), 107–117.

[24] Gavin, Brendan, and Polizzi, Eric. Non-linear eigensolver-based alternative to traditional scf methods. *J. Chem. Phys. 138* (2013), 194101.

[25] Gavin, Brendan, and Polizzi, Eric. Enhancing the performance and robustness of the feast eigensolver. In *High Performance Extreme Computing Conference (HPEC), 2016 IEEE* (2016), IEEE, pp. 1–6.

[26] Gavin, Brendan, and Polizzi, Eric. Krylov eigenvalue strategy using the feast algorithm with inexact system solves. *Numerical Linear Algebra with Applications* (2018), e2188.

[27] Genovese, Luigi, Videau, Brice, Ospici, Matthieu, Deutsch, Thierry, Goedecker, Stefan, and Méhaut, Jean-François. Daubechies wavelets for high performance electronic structure calculations: The bigdft project. *Comptes Rendus Mécanique 339*, 2-3 (2011), 149–164.

[28] Giannozzi, Paolo, Baroni, Stefano, Bonini, Nicola, Calandra, Matteo, Car, Roberto, Cavazzoni, Carlo, Ceresoli, Davide, Chiarotti, Guido L, Cococcioni, Matteo, Dabo, Ismaila, et al. Quantum espresso: a modular and open-source software project for quantum simulations of materials. *Journal of physics: Condensed matter 21*, 39 (2009), 395502.

[29] Gohberg, I., Lancaster, P., and Rodman, L. *Matrix polynomials*, vol. 58 of *Classics in Applied Mathematics*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2009. Reprint of the 1982 original [MR0662418].

[30] Golub, Gene H, and Van Loan, Charles F. *Matrix computations*, vol. 3. JHU Press, 2012.

[31] Golub, Gene H, and Ye, Qiang. Inexact inverse iteration for generalized eigenvalue problems. *BIT Numerical Mathematics 40*, 4 (2000), 671–684.

[32] Gutknecht, Martin H. The unsymmetric lanczos algorithms and their relations to p ade approximation, continued fractions and the qd algorithm. In *Proceedings of the Copper Mountain Conference on Iterative Methods* (1990), vol. 2.

[33] Güttel, S., and Tisseur, F. The nonlinear eigenvalue problem. *Acta Numer. 26* (2017), 1–94.

[34] Güttel, Stefan, Polizzi, Eric, Tang, Ping Tak Peter, and Viaud, Gautier. Zolotarev quadrature rules and load balancing for the feast eigensolver. *SIAM Journal on Scientific Computing 37*, 4 (2015), A2100–A2122.

[35] Güttel, Stefan, Van Beeumen, Roel, Meerbergen, Karl, and Michiels, Wim. NLEIGS: a class of fully rational Krylov methods for nonlinear eigenvalue problems. *SIAM J. Sci. Comput. 36*, 6 (2014), A2842–A2864.

[36] Hadeler, Karl P. Mehrparametrige und nichtlineare eigenwertaufgaben. *Archive for Rational Mechanics and Analysis 27*, 4 (1967), 306–328.

[37] Hernandez, Vicente, Roman, Jose E, and Vidal, Vicente. Slepc: A scalable and flexible toolkit for the solution of eigenvalue problems. *ACM Transactions on Mathematical Software (TOMS) 31*, 3 (2005), 351–362.

[38] Hohenberg, Pierre, and Kohn, Walter. Inhomogeneous electron gas. *Physical review 136*, 3B (1964), B864.

[39] Imakura, Akira, Du, Lei, and Sakurai, Tetsuya. A block Arnoldi-type contour integral spectral projection method for solving generalized eigenvalue problems. *Appl. Math. Lett. 32* (2014), 22–27.

[40] Jarlebring, Elias, and Michiels, Wim. Analyzing the convergence factor of residual inverse iteration. *BIT Numerical Mathematics 51*, 4 (2011), 937–957.

[41] Jarlebring, Elias, and Voss, Heinrich. Rational Krylov for nonlinear eigenproblems, an iterative projection method. *Applications of Mathematics 50*, 6 (2005), 543–554.

[42] Kariya, Takeaki, and Kurata, Hiroshi. *Generalized least squares*. John Wiley & Sons, 2004.

[43] Knyazev, Andrew V, Argentati, Merico E, Lashuk, Ilya, and Ovtchinnikov, Evgueni E. Block locally optimal preconditioned eigenvalue xolvers (blopex) in hypre and petsc. *SIAM Journal on Scientific Computing 29*, 5 (2007), 2224–2239.

[44] Kohn, Walter, and Sham, Lu Jeu. Self-consistent equations including exchange and correlation effects. *Physical review 140*, 4A (1965), A1133.

[45] Kronik, Leeor, Makmal, Adi, Tiago, Murilo L, Alemany, MMG, Jain, Manish, Huang, Xiangyang, Saad, Yousef, and Chelikowsky, James R. Parsec–the pseudopotential algorithm for real-space electronic structure calculations: recent advances and novel applications to nano-structures. *physica status solidi (b) 243*, 5 (2006), 1063–1079.

[46] Lai, Yu-Ling, Lin, Kun-Yi, and Lin, Wen-Wei. An inexact inverse iteration for large sparse eigenvalue problems. *Numerical Linear Algebra with Applications 4*, 5 (1997), 425–437.

[47] Lancaster, P., and Rózsa, P. The spectrum and stability of a vibrating rail supported by sleepers. *Comput. Math. Appl. 31*, 4-5 (1996), 201–213. Selected topics in numerical methods (Miskolc, 1994).

[48] Lanczos, Cornelius. Solution of systems of linear equations by minimized iterations. *J. Res. Nat. Bur. Standards 49*, 1 (1952), 33–53.

[49] Lehoucq, Richard B, Sorensen, Danny C, and Yang, Chao. *ARPACK users' guide: solution of large-scale eigenvalue problems with implicitly restarted Arnoldi methods*, vol. 6. Siam, 1998.

[50] Li, Ruipeng, Xi, Yuanzhe, Vecharynski, Eugene, Yang, Chao, and Saad, Yousef. A thick-restart lanczos algorithm with polynomial filtering for hermitian eigenvalue problems. *SIAM Journal on Scientific Computing 38*, 4 (2016), A2512–A2534.

[51] Mackey, D. Steven, Mackey, Niloufer, and Tisseur, Françoise. Polynomial eigenvalue problems: theory, computation, and structure. In *Numerical algebra, matrix theory, differential-algebraic equations and control theory*. Springer, Cham, 2015, pp. 319–348.

[52] Mehrmann, V., and Watkins, D. Polynomial eigenvalue problems with Hamiltonian structure. *Electron. Trans. Numer. Anal. 13* (2002), 106–118.

[53] Mehrmann, Volker, and Voss, Heinrich. Nonlinear eigenvalue problems: a challenge for modern eigenvalue methods. *GAMM Mitt. Ges. Angew. Math. Mech. 27*, 2 (2004), 121–152 (2005).

[54] Morgan, Ronald. On restarting the arnoldi method for large nonsymmetric eigenvalue problems. *Mathematics of Computation of the American Mathematical Society 65*, 215 (1996), 1213–1230.

[55] Morgan, Ronald B. Computing interior eigenvalues of large matrices. *Linear Algebra and its Applications 154* (1991), 289–309.

[56] Morgan, Ronald B, and Zeng, Min. Harmonic projection methods for large non-symmetric eigenvalue problems. *Numerical linear algebra with applications 5*, 1 (1998), 33–55.

[57] Neumaier, A. Residual inverse iteration for the nonlinear eigenvalue problem. *SIAM journal on numerical analysis 22*, 5 (1985), 914–923.

[58] Notay, Yvan. Convergence analysis of inexact rayleigh quotient iteration. *SIAM Journal on Matrix Analysis and Applications 24*, 3 (2003), 627–644.

[59] Paige, Christopher C, and Saunders, Michael A. Solution of sparse indefinite systems of linear equations. *SIAM journal on numerical analysis 12*, 4 (1975), 617–629.

[60] Parlett, Beresford N. *The symmetric eigenvalue problem*. SIAM, 1998.

[61] Pathria, RK. Statistical mechanics, international series in natural philosophy, 1986.

[62] Polizzi, Eric. Feast eigenvalue solver. `http://www.feast-solver.org`. Accessed: 2017-08-13.

[63] Polizzi, Eric. Density-matrix-based algorithm for solving eigenvalue problems. *Phys. Rev. B 79* (2009), 115112.

[64] Robbé, Mickaël, Sadkane, Miloud, and Spence, Alastair. Inexact inverse subspace iteration with preconditioning applied to non-hermitian eigenvalue problems. *SIAM Journal on Matrix Analysis and Applications 31*, 1 (2009), 92–113.

[65] Roweis, Sam T, and Saul, Lawrence K. Nonlinear dimensionality reduction by locally linear embedding. *science 290*, 5500 (2000), 2323–2326.

[66] Ruhe, Axel. Algorithms for the nonlinear eigenvalue problem. *SIAM J. Numer. Anal. 10*, 4 (1973), 674–689.

[67] Ruhe, Axel. Rational krylov sequence methods for eigenvalue computation. *Linear Algebra and its Applications 58* (1984), 391–405.

[68] Ruhe, Axel. The rational Krylov algorithm for nonlinear matrix eigenvalue problems. *Journal of Mathematical Sciences 114*, 6 (2003), 1854–1856.

[69] Runge, Erich, and Gross, Eberhard KU. Density-functional theory for time-dependent systems. *Physical Review Letters 52*, 12 (1984), 997.

[70] Saad, Youcef. Chebyshev acceleration techniques for solving nonsymmetric eigenvalue problems. *Mathematics of Computation 42*, 166 (1984), 567–588.

[71] Saad, Youcef. *Numerical Methods for Large Eigenvalue Problems*, vol. 158. SIAM, 1992.

[72] Saad, Youcef, and Schultz, Martin H. Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on scientific and statistical computing 7*, 3 (1986), 856–869.

[73] Saad, Yousef. *Iterative methods for sparse linear systems*. SIAM, 2003.

[74] Saad, Yousef. Analysis of subspace iteration for eigenvalue problems with evolving matrices. *SIAM Journal on Matrix Analysis and Applications 37*, 1 (2016), 103–122.

[75] Sakurai, Jun John, and Commins, Eugene D. Modern quantum mechanics, revised edition, 1995.

[76] Sakurai, Tetsuya, and Sugiura, Hiroshi. A projection method for generalized eigenvalue problems using numerical integration. In *Proceedings of the 6th Japan-China Joint Seminar on Numerical Mathematics (Tsukuba, 2002)* (2003), vol. 159, pp. 119–128.

[77] Sakurai, Tetsuya, and Tadano, Hiroto. CIRR: a Rayleigh-Ritz type method with contour integral for generalized eigenvalue problems. *Hokkaido Math. J. 36*, 4 (2007), 745–757.

[78] Schenk, Olaf, Gärtner, K, Karypis, G, Röllin, S, and Hagemann, M. Pardiso solver project. *URL: http://www. pardiso-project. org* (2010).

[79] Schofield, Grady, Chelikowsky, James R, and Saad, Yousef. A spectrum slicing method for the kohn–sham problem. *Computer Physics Communications 183*, 3 (2012), 497–505.

[80] Schölkopf, Bernhard, Smola, Alexander, and Müller, Klaus-Robert. Nonlinear component analysis as a kernel eigenvalue problem. *Neural computation 10*, 5 (1998), 1299–1319.

[81] Shao, Zhi-an, Porod, Wolfgang, Lent, Craig S, and Kirkner, David J. An eigenvalue method for open-boundary quantum transmission problems. *J. Appl. Phys. 78*, 4 (1995), 2177–2186.

[82] Siegert, A. J. F. On the derivation of the dispersion formula for nuclear reactions. *Phys. Rev. 56* (Oct 1939), 750–752.

[83] Sima, Diana M, Van Huffel, Sabine, and Golub, Gene H. Regularized total least squares based on quadratic eigenvalue problem solvers. *BIT Numerical Mathematics 44*, 4 (2004), 793–812.

[84] Simoncini, Valeria, and Eldén, Lars. Inexact rayleigh quotient-type methods for eigenvalue computations. *BIT Numerical Mathematics 42*, 1 (2002), 159–182.

[85] Sleijpen, Gerard LG, Booten, Albert GL, Fokkema, Diederik R, and Van der Vorst, Henk A. Jacobi-davidson type methods for generalized eigenproblems and polynomial eigenproblems. *BIT Numerical Mathematics 36*, 3 (1996), 595–633.

[86] Sleijpen, Gerard LG, and Fokkema, Diederik R. Bicgstab (l) for linear equations involving unsymmetric matrices with complex spectrum. *Electronic Transactions on Numerical Analysis 1*, 11 (1993), 2000.

[87] Sleijpen, Gerard LG, and Van der Vorst, Henk A. A jacobi–davidson iteration method for linear eigenvalue problems. *SIAM review 42*, 2 (2000), 267–293.

[88] Sorensen, Danny C. Implicit application of polynomial filters in a k-step arnoldi method. *Siam journal on matrix analysis and applications 13*, 1 (1992), 357–385.

[89] Stathopoulos, Andreas, and McCombs, James R. Primme: preconditioned iterative multimethod eigensolvermethods and software description. *ACM Transactions on Mathematical Software (TOMS) 37*, 2 (2010), 21.

[90] Stathopoulos, Andreas, and Saad, Yousef. Restarting techniques for the (jacobi-) davidson symmetric eigenvalue methods. *Electron. Trans. Numer. Anal 7* (1998), 163–181.

[91] Stewart, Gilbert W. *Matrix algorithms volume 2: eigensystems*, vol. 2. Siam, 2001.

[92] T., J. Asakura, Sakurai, Tadano, H., Ikegami, T., and Kimura, K. A numerical method for polynomial eigenvalue problems using contour integral. *Jpn. J. Ind. Appl. Math. 27*, 1 (2010), 73–90.

[93] Tang, Ping Tak Peter, Kestyn, James, and Polizzi, Eric. A new highly parallel non-hermitian eigensolver. In *Proceedings of the High Performance Computing Symposium* (2014), Society for Computer Simulation International, p. 1.

[94] Tang, Ping Tak Peter, and Polizzi, Eric. Feast as subspace iteration accelerated by approximate spectral projection. *SIAM Journal on Matrix Analysis and Applications 35* (2014), 354390.

[95] Tenenbaum, Joshua B, De Silva, Vin, and Langford, John C. A global geometric framework for nonlinear dimensionality reduction. *science 290*, 5500 (2000), 2319–2323.

[96] Tisseur, F., and Higham, N. J. Structured pseudospectra for polynomial eigenvalue problems, with applications. *SIAM J. Matrix Anal. Appl. 23*, 1 (2001), 187–208.

[97] Tisseur, Françoise, and Meerbergen, Karl. The quadratic eigenvalue problem. *SIAM Rev. 43*, 2 (2001), 235–286.

[98] Trefethen, Lloyd N. Pseudospectra of matrices. *Numerical analysis 91* (1991), 234–266.

[99] Trefethen, Lloyd N, and Bau III, David. *Numerical linear algebra*, vol. 50. Siam, 1997.

[100] Trefethen, Lloyd N, and Weideman, JAC. The exponentially convergent trapezoidal rule. *SIAM Review 56*, 3 (2014), 385–458.

[101] Van der Vorst, Henk A. Bi-cgstab: A fast and smoothly converging variant of bi-cg for the solution of nonsymmetric linear systems. *SIAM Journal on scientific and Statistical Computing 13*, 2 (1992), 631–644.

[102] Van der Vorst, Henk A. *Iterative Krylov methods for large linear systems*, vol. 13. Cambridge University Press, 2003.

[103] Voss, H. An Arnoldi method for nonlinear eigenvalue problems. *BIT 44*, 2 (2004), 387–401.

[104] Voss, H. Nonlinear Eigenvalue Problems - Chapter 60. In *Handbook of Linear Algebra, Second Edition*, L. Hogben, Ed., Discrete Mathematics and its Applications. Chapman & Hall/CRC, Boca Raton, FL, 2013, pp. 1063–1086.

[105] Voss, Heinrich. A new justification of the jacobi–davidson method for large eigenproblems. *Linear algebra and its applications 424*, 2-3 (2007), 448–455.

[106] Wilkinson, James Hardy. *Rounding errors in algebraic processes*. Courier Corporation, 1994.

[107] Xue, Fei, and Elman, Howard C. Fast inexact subspace iteration for generalized eigenvalue problems with spectral transformation. *Linear Algebra and its Applications 435*, 3 (2011), 601–622.

[108] Yokota, S., and Sakurai, T. A projection method for nonlinear eigenvalue problems using contour integrals. *JSIAM Lett. 5* (2013), 41–44.

[109] Zhou, Yunkai. Studies on jacobi–davidson, rayleigh quotient iteration, inverse iteration generalized davidson and newton updates. *Numerical Linear Algebra with Applications 13*, 8 (2006), 621–642.

[110] Zhou, Yunkai, Saad, Yousef, Tiago, Murilo L, and Chelikowsky, James R. Self-consistent-field calculations using chebyshev-filtered subspace iteration. *Journal of Computational Physics 219*, 1 (2006), 172–184.