Spring 2015

Parallel symmetric eigenvalue problem solvers

Alicia Marie Klinvex
Purdue University

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Alicia Marie Klinvex

Entitled
Parallel Symmetric Eigenvalue Problem Solvers

For the degree of Doctor of Philosophy

Is approved by the final examining committee:

Ahmed Sameh Chair	Rudolf Eigenmann
Ananth Grama	
Alex Pothen	
Robert Skeel	

To the best of my knowledge and as understood by the student in the Thesis/Dissertation Agreement, Publication Delay, and Certification Disclaimer (Graduate School Form 32), this thesis/dissertation adheres to the provisions of Purdue University's "Policy of Integrity in Research" and the use of copyright material.

Approved by Major Professor(s): Ahmed Sameh

Approved by: Sunil Prabhakar 4/23/2015
Head of the Departmental Graduate Program Date

PARALLEL SYMMETRIC EIGENVALUE PROBLEM SOLVERS

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Alicia Marie Klinvex

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

May 2015

Purdue University

West Lafayette, Indiana

To my grandparents,
Fred and Joy McClemens

ACKNOWLEDGMENTS

First and foremost, I am grateful to my advisor Dr. Ahmed Sameh for his guidance and support all throughout my graduate career. During my six years as his student, he was never too busy to help me or any other student. Similarly, I would like to thank the other members of my advisory committee, Professors Ananth Grama, Robert Skeel, Alex Pothen, and Rudolf Eigenmann who took the time to review my work and provide meaningful commentary. I would also like to thank Dr. Faisal Saied for his tireless assistance during my first years of graduate school. Debugging somebody else's MPI code is an immensely frustrating experience, but he would regularly stay late at the office to assist me anyway. I am indebted to Zhengyi Zhang and Vasilis Kalantzis, who were always willing to help out any way they could in spite of having their own research to work on.

I am also quite grateful to Mike Heroux and Mike Parks for allowing me to spend a summer at Sandia National Laboratories as an intern, and to Karen Devine, Heidi Thornquist, Rich Lehoucq, and Erik Boman for their guidance while I was there. Mark Hoemmen deserves special thanks for helping me with the software engineering aspects of creating a Trilinos-based implementation of TraceMin. I would also like to thank the PETSc and SLEPc development teams for the helpful email correspondance regarding how to install and use their respective packages. I am grateful to Intel Corporation, specifically David Kuck and Mallick Arigapudi, for allowing me to use their computing resources to conduct my tests, and to the Army Research Office, which funded several of my trips to various conferences under grant number 7W911NF-11-1-0401.

Additionally, I would like to thank Dr. William Gorman for answering my many questions and helping with the formatting of my dissertation, and Tammy Muthig (as well as the other employees of the department's Business Office) for handling my numerous financial aid issues. I am also quite grateful to Professors Ron McCarty, Gary Walker, Meng Su, Charles Burchard, and Blair Tuttle at Penn State. Apart from providing encouragement and advice, they all employed me so that I could get research, tutoring, and mentoring experience as an undergraduate. Last but not least, I thank my family for their love and support.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

ABSTRACT

Klinvex, Alicia Marie Ph.D., Purdue University, May 2015. Parallel Symmetric Eigenvalue Problem Solvers. Major Professors: Ahmed Sameh.

Sparse symmetric eigenvalue problems arise in many computational science and engineering applications: in structural mechanics, nanoelectronics, and spectral reordering, for example. Often, the large size of these problems requires the development of eigensolvers that scale well on parallel computing platforms. In this dissertation, we describe two such eigensolvers, TraceMin and TraceMin-Davidson. These methods are different from many other eigensolvers in that they do not require accurate linear solves to be performed at each iteration in order to find the smallest eigenvalues and their associated eigenvectors. After introducing these closely related eigensolvers, we discuss alternative methods for solving the saddle point problems arising at each iteration, which can improve the overall running time. Additionally, we present TraceMin-Multisectioning, a new TraceMin implementation geared towards finding large numbers of eigenpairs in any given interval of the spectrum. We conclude with numerical experiments comparing our trace-minimization solvers to other popular eigensolvers (such as Krylov-Schur, LOBPCG, Jacobi-Davidson, and FEAST), establishing the competitiveness of our methods.

1 INTRODUCTION

Many applications in science and engineering give rise to symmetric eigenvalue problems of the form

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

where the matrices A and B are sparse and often quite large. We seek the smallest magnitude eigenvalues of a given matrix pencil (A, B) along with their associated eigenvectors. Computing the smallest eigenvalues is more difficult than computing the largest, because it often necessitates the accurate solution of linear systems at each iteration. This can be problematic for direct solvers when the matrices are large, because the level of fill-in may be too large for such factorizations to be possible. Alternatively, they may require the use of strong preconditioners with limited scalability. In some applications, the matrices are not even made explicitly available, which makes preconditioning difficult and factorization impossible. In this dissertation, we present several eigensolvers that do not rely on accurate linear solves, which we refer to as trace-minimization eigensolvers.

First, we discuss a few sample application areas that give rise to sparse symmetric eigenvalue problems. One application area is the modeling of acoustic fields in moving vehicles, which is governed by the lossless wave equation. By applying a finite element discretization, we obtain a generalized eigenvalue problem where both the stiffness and mass matrices are ill-conditioned. We are interested in computing all eigenvalues in a given interval along with their associated eigenvectors.

Another problem of interest is the Anderson model of localization, which models electron transport in a random lattice. To examine this behavior, we must solve the time-independent Schrödinger equation, a standard eigenvalue problem. The eigenvalues of that matrix represent potential energy, and the eigenvectors give us the probability of an electron residing at a particular site; we are interested in the lowest potential energies, meaning the eigenvalues closest to 0. If the magnitude of each element of the eigenvector is approximately equal, then the material conducts. Otherwise, the material does not. This problem is difficult because the desired eigenvalues are interior, which are notably harder to obtain than extreme eigenvalues.

The last application area we disucss is spectral reordering. Unweighted bandwidth reducing reorderings are important because they can reduce the cost of parallel matrix-vector multiplications by bringing the elements of the matrix toward the diagonal, resulting in less communication between MPI processes. Weighted reorderings can be useful in constructing banded preconditioners, because they bring the large elements of a matrix toward the diagonal. To compute the Fiedler vector for spectral reordering, we must solve a standard eigenvalue problem where A is symmetric positive semidefinite, and the null space of A is known. This problem can be difficult for some eigensolvers because A is singular.

After providing motivation for the development of scalable sparse symmetric eigensolvers, we discuss an important kernel in the trace-minimization eigensolvers: the solution of saddle point problems of the form

$$\begin{bmatrix} A & BY \\ Y^T B & 0 \end{bmatrix} \begin{bmatrix} \Delta \\ L \end{bmatrix} = \begin{bmatrix} AY \\ 0 \end{bmatrix} \tag{1.2}$$

where Y_k is a tall dense matrix with a very small number of columns. We present three types of methods for solving that linear system, then discuss under which circumstances each should be used.

One way to solve this problem is by using a projected Krylov method to solve the equivalent linear system

$$PAP\Delta = PAY \tag{1.3}$$

where

$$P = I - BY \left(Y^T B^2 Y\right)^{-1} Y^T B \tag{1.4}$$

projects onto the space orthogonal to BY. Another method of solving the saddle point problem is by forming the Schur complement

$$S = -Y^T B A^{-1} B Y \tag{1.5}$$

After we obtain the Schur complement (which can be inexact if we used a Krylov method to determine $A^{-1}BY$), we may construct the solution $\Delta = Y + A^{-1}BYS^{-1}$. The last method we discuss is the use of block preconditioned Krylov methods. We may look at our original saddle point problem (equation 1.2) as a linear system $\mathscr{A}\mathscr{X} = \mathscr{F}$ and use a Krylov subspace method on the entire problem. We can precondition this linear system in a variety of ways. One such preconditioner is

$$\mathscr{M} = \begin{bmatrix} M & 0 \\ 0 & \hat{S} \end{bmatrix} \tag{1.6}$$

where M is a preconditioner approximating A, and $\hat{S} = -Y^T B M^{-1} B Y$.

After exploring how to solve the saddle point problems arising at each iteration of the trace minimization eigensolvers, we describe two such solvers: TraceMin and TraceMin-Davidson. As the name suggests, these eigensolvers transform the problem of computing the desired eigenpairs into the equivalent constrained minimization problem

$$\min_{Y^T BY=I} \operatorname{trace}\left(Y^T AY\right) \tag{1.7}$$

The solution to this problem is the set of eigenvectors corresponding to the eigenvalues of smallest magnitude. At each iteration of our trace-minimization eigensolver, we compute an update Δ_k to our current approximate eigenvectors Y_k such that $\Delta_k \perp_B Y_k$ and

$$\operatorname{trace}\left((Y_k - \Delta_k)^T A (Y_k - \Delta_k)\right) < \operatorname{trace}\left(Y_k^T A Y_k\right) \tag{1.8}$$

When we solve the corresponding constrained minimization problem

$$\min_{\Delta_k \perp_B Y_k} \operatorname{trace}\left((Y_k - \Delta_k)^T A \, (Y_k - \Delta_k) \right) \tag{1.9}$$

using Lagrange multipliers, we end up with the saddle point problem previously discussed. The only difference between TraceMin and TraceMin-Davidson is that TraceMin extracts its Ritz vectors Y_k from a subspace of constant dimension, whereas TraceMin-Davidson uses expanding subspaces. These algorithms are explained in detail in their respective chapters.

TraceMin has global linear convergence, the rate of which is based on both the distribution of eigenvalues and the constant subspace dimension s. In the TraceMin chapter, we present small test cases that show how the behavior of TraceMin changes when you modify various parameters such as the subspace dimension or tolerance of the Krylov method. We also explain how the convergence rate can be improved by using dynamic origin shifts, which are determined by the Ritz values of the matrix pencil. We conclude with a discussion of the relationship between TraceMin and simultaneous iteration. If both methods solve the linear systems arising at each iteration exactly (using a direct method), the methods are equivalent. However, we show that TraceMin is more robust and tolerates inexact solves with very little precision better than simultaneous iteration.

In the TraceMin-Davidson chapter, we discuss how the method differs from TraceMin through the use of expanding subspaces. We also present a small experiment showing the effect of the block size on finding eigenvalues with a multiplicity greater than 1. Additionally, we describe what harmonic Ritz extraction is and how it can help when computing interior eigenpairs.

After describing the theory of these eigensolvers, we describe our parallel implementations of these methods in solving different types of problems. First, we discuss our publically available Trilinos implementations, which are designed to compute a small number of eigenpairs of smallest magnitude. We then explain how spectral transformations can be used to compute the largest eigenvalues or the eigenvalues nearest a given shift, and how spectrum folding can allow eigensolvers which are designed for the computation of extreme eigenpairs to compute interior ones successfully. In addition, we describe the parallel kernels required by our code.

The next sections describe our Fortran-based implementations of sampling and multisectioning. In the case of sampling, we are interested in computing the eigenvalues closest to a large set of shifts; in multisectioning (or spectrum slicing), we are interested in computing all eigenvalues in a given interval. This interval often contains a large number of eigenvalues. We present a multisectioning algorithm loosely based on adaptive quadrature which divides the large global interval of interest into many subintervals which can be processed independently. Our method performs both the interval subdivision step and the eigensolver steps in parallel and features dynamic load balancing for improved scalability.

After we have thoroughly explored TraceMin and TraceMin-Davidson, we describe several other eigensolvers which compete against our implementations in the numerical experiments section. Arnoldi, Lanczos, and Krylov-Schur are very similar methods, all of which require the accurate solution of linear systems at each iteration; they are analogous to TraceMin-Davidson, if we use the Schur-complement method to solve the saddle point problem at each iteration. The Locally Optimal Block Preconditioned Conjugate Gradient method avoids solving linear systems entirely, but it can fail if not given a strong preconditioner. Jacobi-Davidson is theoretically similar to TraceMin-Davidson, except that it uses a more aggressive shifting strategy which can cause it to miss the smallest eigenpairs or converge very slowly. The Riemannian Trust Region method is very closely related to TraceMin, but it uses the exact Hessian in solving the constrained minimization problem whereas TraceMin uses a cheap approximation. FEAST is a contour integration based eigensolver which requires both an interval of interest and an estimate of the number of eigenvalues that interval contains.

Finally, we present comparisons between our methods and those of the popular eigensolver packages Anasazi (of Sandia's Trilinos library), SLEPc, and FEAST, establishing the robustness and parallel scalability of TraceMin.

2 MOTIVATING APPLICATIONS

In this section, we justify the need for a robust and parallel sparse symmetric eigenvalue problem solver such as TraceMin by presenting several application areas which give rise to large sparse symmetric eigenvalue problems.

2.1 Automotive engineering

Modeling acoustic fields in moving vehicles generally uses coupled systems of partial differential equations (PDEs). The systems resulting from the discretization of these PDEs tend to be extremely large and ill-conditioned.

The lossless wave equation in air is given by

$$\Delta p - \frac{1}{c^2}\frac{\delta^2 p}{\delta t^2} = 0 \tag{2.1}$$

where p represents the pressure and c the speed of sound; for a derivation of this equation, please see [1]. Neumann boundary conditions are given by

$$\frac{\delta p}{\delta \nu} = -\frac{r}{\rho_0^2 c^2}\frac{\delta p}{\delta t} \tag{2.2}$$

where ρ represents the density, r the damping properties of the material, and ν the outer normal. We may apply a finite element discretization to obtain the following equation for the fluid,

$$M_f \ddot{p}_d + D_f \dot{p}_d + K_f p_d + D_{sf} \ddot{u}_d = 0 \tag{2.3}$$

where M_f is a spd mass matrix, K_f is a spd stiffness matrix, D_f is a spsd damping matrix, D_{sf} is a spd mass matrix representing the fluid structure coupling, and u represents the vector of displacements.

The discrete finite element model for the vibration of the structure is

$$M_s\ddot{u}_d + D_s\dot{u}_d + K_s u_d - D_{sf}^T p_d = f_e \tag{2.4}$$

with M_s and K_s spd, D_s spsd, and f_e the external load. If we combine equations 2.3 and 2.4, we obtain

$$\begin{bmatrix} M_s & 0 \\ D_{sf} & M_f \end{bmatrix}\begin{bmatrix} \ddot{u}_d \\ \ddot{p}_d \end{bmatrix} + \begin{bmatrix} D_s & 0 \\ 0 & D_f \end{bmatrix}\begin{bmatrix} \dot{u}_d \\ \dot{p}_d \end{bmatrix} + \begin{bmatrix} K_s(\omega) & -D_{sf}^T \\ 0 & K_f \end{bmatrix}\begin{bmatrix} u_d \\ p_d \end{bmatrix} = \begin{bmatrix} f_s \\ 0 \end{bmatrix} \tag{2.5}$$

We then perform a Fourier ansatz

$$\begin{bmatrix} u_d \\ p_d \end{bmatrix} = \begin{bmatrix} \hat{u} \\ \hat{p} \end{bmatrix} e^{i\omega t}, f_s = \hat{f}e^{i\omega t} \tag{2.6}$$

to obtain the following frequency dependent linear system

$$\left(-\omega^2\begin{bmatrix} M_s & 0 \\ D_{sf} & M_f \end{bmatrix} + i\omega\begin{bmatrix} D_s & 0 \\ 0 & D_f \end{bmatrix} + \begin{bmatrix} K_s(\omega) & -D_{sf}^T \\ 0 & K_f \end{bmatrix}\right)\begin{bmatrix} \hat{u}(\omega) \\ \hat{p}(\omega) \end{bmatrix} = \begin{bmatrix} \hat{f}(\omega) \\ 0 \end{bmatrix} \tag{2.7}$$

We may also write this as a symmetric problem for nonzero frequencies.

$$\left(-\omega^2\begin{bmatrix} M_s & 0 \\ 0 & M_f \end{bmatrix} + i\omega\begin{bmatrix} D_s & iD_{sf}^T \\ iD_{sf} & D_f \end{bmatrix} + \begin{bmatrix} K_s(\omega) & 0 \\ 0 & K_f \end{bmatrix}\right)\begin{bmatrix} \hat{u}(\omega) \\ \omega^{-1}\hat{p}(\omega) \end{bmatrix} = \begin{bmatrix} \hat{f}(\omega) \\ 0 \end{bmatrix} \tag{2.8}$$

Although these systems have very large dimensions, we are typically only interested in the low frequencies associated with the eigenvalues in the neighborhood of zero of the following symmetric matrix function

$$Q(\omega) = -\omega^2 M + i\omega D + K \tag{2.9}$$

where K's nonlinear dependency on the frequency is ignored. In the absence of damping, equation 2.9 gives rise to the following generalized eigenvalue problem

$$Kx = \lambda Mx \tag{2.10}$$

where, in exact arithmetic M and K are spd, but M is singular to working precision in floating-point arithmetic due to the fact that rotational masses are omitted [1].

2.2 Condensed matter physics

In 1958, P.W. Anderson proposed a model for electron transport in a random lattice [2]. Although it was later discovered that Anderson localization may occur for any wave propagating through a disordered medium [3], we focus on the model as it applies to conductivity.

In this model, we have an array of sites called a lattice. These sites are occupied by entities such as atoms. The basic technique is to place a single electron in the lattice and study the resulting behavior of the wave function. The wave function tells us the probability of finding an electron at a particular site. If the probability of finding an electron at certain sites is practically zero, Anderson localization occurs as in figure 2.1.

To examine this phenomenon, we solve the time-independent Schroedinger equation

$$E\Psi = \widehat{H}\Psi \tag{2.11}$$

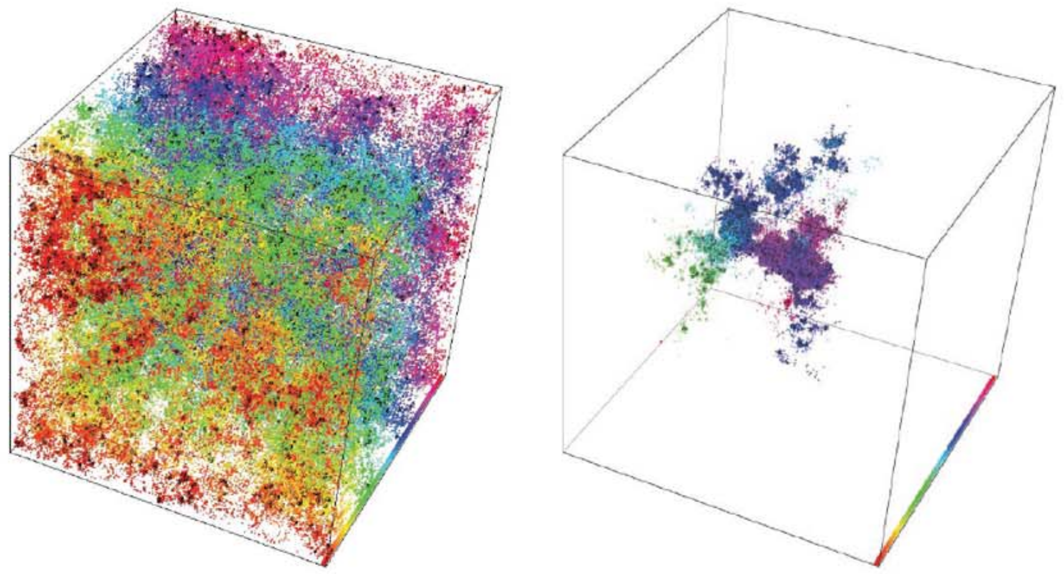


Figure 2.1.: Behavior of an externalized (left) and localized (right) wavefunction for the three-dimensional Anderson model with periodic boundary conditions at $E = 0$ with $N = 100^3$ and $W = 12$ and 21 respectively [4]

with Hamiltonian

$$\left(\widehat{H}\phi\right)(j) = E_j\phi(j) + \sum_{k\neq j} V(|k-j|)\,\phi(k) \tag{2.12}$$

The goal is to find the stationary states Ψ with low energy E, i.e. to find the eigenvalues closest to 0 and their associated eigenvectors.

The first term of the Hamiltonian accounts for the probability of an electron at a particular site staying there, and it is based on the randomly assigned energy at that site, E_j. The second term accounts for the probability of the electron hopping to an adjacent site and is based on the interaction term $V(r)$. For simplicity, we may choose

$$V(|r|) = \begin{cases} 1 & \text{if } |r| = 1 \\ 0 & \text{otherwise} \end{cases} \tag{2.13}$$

This gives us a matrix with the same structure as the seven-point central difference approximation to the three-dimensional Poisson equation on the unit cube with periodic boundary conditions.

The difficulty of this problem arises from the random entries on the diagonal and the large cluster of eigenvalues around 0. Each E_j is taken from a uniform distribution in $[-W/2, +W/2]$ with some $W \in [1, 30]$, meaning the Hamiltonian matrix will likely be symmetric indefinite. This W changes the behavior of the material in the following way:

- If $W << 16.5$, the eigenvectors are extended and the material will be a conductor.
- If $W >> 16.5$, all eigenvectors are localized and the material will be an insulator.
- $W = W_c = 16.5$ is a critical value where the extended states around $E = 0$ vanish and no current can flow.

To numerically distinguish between these three cases, we must look at a series of many large problems of order 10^6 to 10^8 with various random diagonals [5].

2.3 Spectral reordering

Parallel sparse matrix-vector multiplications $y = Ax$ can be very expensive operations due to low data locality. The cost of such operations is based on the sparsity pattern of the matrix A. General sparse matrices can require collective communication between *all* MPI processes, which is very undesirable when running on a large number of nodes. However, if A is banded, we only require point-to-point communication between nearest neighbors. As a result, we may wish to permute the elements of A to gain a more favorable sparsity pattern, keeping in mind that the cost of computing this permutation will be amortized over a large number of matrix-vector multiplications. One method of obtaining this permutation is by computing the Fiedler vector [6]. Computing the Fiedler vector of an unweighted graph produces a bandwidth-reducing reordering, whereas computing the Fiedler vector of a weighted graph produces a reordering which brings large elements toward the diagonal. Bandwidth-reducing reorderings are meant to reduce the cost of a matrix-vector product, and weighted spectral reorderings can be part of an effective preconditioning strategy; after bringing the large elements toward the diagonal, we can extract a band from our reordered matrix to be used as a preconditioner. This preconditoner could be applied by a scalable banded solver such as SPIKE [7,8].

In this case, we are interested in the eigenvector corresponding to the smallest nonzero eigenvalue of a standard eigenvalue problem $Lx = \lambda x$, where L is the graph Laplacian. Assuming D is the degree matrix of A and J is the adjacency matrix, $L = D - J$. The graph Laplacian L is symmetric positive semi-definite. If the graph consists of only one strongly connected component, the graph Laplacian A's null space is exactly one vector, the vector of all 1s[1]. If the graph consists of multiple components, it can be split up into many smaller eigenproblems, one per strongly connected component, and these problems can be solved independently.

[1]We assume for simplicity that A was not normalized.

3 PARALLEL SADDLE POINT SOLVERS

The goal of this chapter is the solution of the following saddle point problem, which arises in every TraceMin iteration.

$$\begin{bmatrix} A & BY \\ Y^T B & 0 \end{bmatrix} \begin{bmatrix} \Delta \\ L \end{bmatrix} = \begin{bmatrix} AY \\ 0 \end{bmatrix} \tag{3.1}$$

A is symmetric, Y has many more rows than columns and is assumed to have full column rank. We also know that $Y^T BY = I$.

We will examine several ways of solving linear systems with this special structure on parallel architectures.

3.1 Using a projected Krylov method

This is the method originally used in the 1982 implementation of a basic trace minimization eigensolver [9]. Solving the system (3.1) is equivalent to solving the following linear system

$$PAP\Delta = PAY \tag{3.2}$$

where

$$P = I - BY \left(Y^T B^2 Y\right)^{-1} Y^T B \tag{3.3}$$

projects onto the space B-orthogonal to Y. Since this linear system is consistent, we can use a Krylov subspace method to solve it (even though our operator PAP is singular).

If we choose our initial iterate $\Delta_0 \perp_B Y$, applying the symmetric operator PAP to a vector (or set of vectors) at each iteration is equivalent to applying PA. That means we can use a symmetric Krylov method such as the conjugate gradient method or MINRES and still only require one projection [10, 11].

3.2 Forming the Schur complement

By performing block Gaussian elimination on equation 3.1, we obtain the following result.

$$\Delta = Y + ZS^{-1} \tag{3.4}$$

where $Z = A^{-1}BY$ and $S = -Y^TBZ$ is the Schur complement. We can solve $AZ = BY$ approximately using a Krylov method to obtain the inexact Schur complement $\hat{S} = -BY^T\hat{Z}$. Once we have $\hat{Z}$ and $\hat{S}$, we can compute Δ using a small dense solve and a vector addition.

3.3 Block preconditioned Krylov methods

Another alternative is to use a Krylov subspace method on the entire problem, meaning we have the operator

$$\mathscr{A} = \begin{bmatrix} A & BY \\ Y^TB & 0 \end{bmatrix} \tag{3.5}$$

We can precondition this operator in a variety of ways. If we use the preconditioner

$$\mathscr{M} = \begin{bmatrix} A & 0 \\ 0 & -S \end{bmatrix} \tag{3.6}$$

where $S = -Y^TBA^{-1}BY$ is the Schur complement, then our preconditioned matrix $\mathscr{M}^{-1}\mathscr{A}$ has at most four distinct eigenvalues [12]. Therefore, we would converge in at most four iterations of MINRES in exact arithmetic. However, each iteration would involve the accurate solution of linear systems involving A, which can be very expensive.

Alternatively, we can replace A by a preconditioner M to obtain

$$\mathscr{M} = \begin{bmatrix} M & 0 \\ 0 & -\hat{S} \end{bmatrix} \tag{3.7}$$

where $\hat{S} = -Y^T B M^{-1} B Y$. Since Y is very narrow, we can compute the matrix $\hat{S}$ explicitly and replicate it across all MPI processes. The application of this preconditioner $\mathscr{M}$ only involves an application of the preconditioner M and a small dense solve with $\hat{S}$.

Note that this is not the only possible preconditioning strategy for this saddle point problem. Instead of using that block diagonal preconditioner, we could use a block triangular preconditioner such as

$$\mathscr{M} = \begin{bmatrix} M & BY \\ 0 & -\hat{S} \end{bmatrix} \tag{3.8}$$

but that would prevent us from using a symmetric solver such as MINRES to solve linear systems with $\mathscr{A}$. We could also choose to do constraint preconditioning with

$$\mathscr{M} = \begin{bmatrix} M & BY \\ Y^T B & 0 \end{bmatrix} \tag{3.9}$$

as in [13].

Equation 3.1 is equivalent to the following

$$\begin{bmatrix} A & BY \\ -Y^T B & 0 \end{bmatrix} \begin{bmatrix} \Delta \\ L \end{bmatrix} = \begin{bmatrix} AY \\ 0 \end{bmatrix} \tag{3.10}$$

The operator of (3.1) is symmetric indefinite; the operator of (3.10) is nonsymmetric, but all eigenvalues will be on one side of the imaginary axis. We could use a Hermitian/Skew-Hermitian splitting based preconditioner on this problem as in [14, 15].

Table 3.1: Comparison of saddle point solvers

	Projected Krylov	Schur complement	Block preconditioning
Strictly enforces the condition $\Delta \perp_B Y$?	yes	no	no
Application of the operator requires an inner product?	yes	no	yes
Capable of using preconditioned MINRES?	no	yes	yes
Capable of using a direct solver?	no	yes	no

3.4 Which method to choose

All of these methods have been incorporated into Sandia's publicly available TraceMin code. Each of the methods has its own unique advantages and disadvantages, summarized in table 3.1.

In short, we recommend the following strategy:

- If you want to factor your matrix A, form the Schur complement. Note that TraceMin does not generally require accurate solutions of linear systems involving A, so this can be overkill.

- If you want to use a preconditioner $M \approx A$, choose the block diagonal preconditioning method. Again, since TraceMin does not require accurate solutions of linear systems involving A, it should not need a strong preconditioner. Preconditioning the projected-Krylov solver does not generally perform well because it requires solutions of nonsymmetric linear systems, which are considerably more expensive than the symmetric case. Forming the inexact Schur complement is another possibility, but it tends to perform poorly without a strict tolerance because the linear system being solved at each iteration is completely disconnected from the requirement that $\Delta \perp_B Y$.

- Otherwise, use the projected Krylov method.

4 TRACEMIN

The generalized eigenvalue problem considered here is given by

$$Ax = \lambda Bx \tag{4.1}$$

where A, B are $n \times n$ very large, sparse, and symmetric, with B being positive definite, and one is interested in obtaining a few eigenvalues $p \ll n$ of smallest magnitude and their associated eigenvectors.

4.1 Derivation of TraceMin

TraceMin is based on the following theorem, which transforms the problem of solving equation (4.1) into a constrained minimization problem.

Theorem 4.1.1 *[9, 16] Let A and B be symmetric $n \times n$ matrices with B positive definite, and Y^* the set of all $n \times p$ matrices Y for which $Y^T BY = I_p$. Then*

$$\min_{Y \in Y^*} \operatorname{tr}(Y^T AY) = \sum_{i=1}^{p} \lambda_i \tag{4.2}$$

where $\lambda_1 \leq \lambda_2 \leq \cdots \lambda_p < \lambda_{p+1} \leq \cdots \leq \lambda_n$ are the eigenvalues of problem (4.1).

The block of vectors Y which solves the constrained minimization problem is the set of eigenvectors corresponding to the eigenvalues of smallest magnitude. We now discuss how TraceMin is derived from this observation.

Let Y be a set of vectors approximating the eigenvectors of interest. At each TraceMin iteration, we wish to construct Y_{k+1} from Y_k such that $\operatorname{tr}\left(Y_{k+1}^T AY_{k+1}\right) < \operatorname{tr}\left(Y_k^T AY_k\right)$. Consequently, Y_{k+1} is a better approximation of the eigenvectors than

Y_k. The iterate Y_k is corrected using the $n \times p$ matrix Δ_k, which is the solution of the following constrained optimization problem,

$$\begin{aligned} \text{minimize} \quad & \operatorname{tr}(Y_k - \Delta_k)^T A (Y_k - \Delta_k), \\ \text{subject to} \quad & Y_k^T B \Delta_k = 0 \end{aligned} \tag{4.3}$$

If A is symmetric positive definite, this is equivalent to solving the p independent problems

$$\begin{aligned} \text{minimize} \quad & (y_{k,i} - d_{k,i})^T A (y_{k,i} - d_{k,i}), \\ \text{subject to} \quad & y_{k,i}^T B \Delta_k = 0 \end{aligned} \tag{4.4}$$

where $y_{k,i}$ is the i-th column of Y_k.

Figure 4.1 presents a graphical illustration of how TraceMin works for the 3x3 matrix

$$\begin{bmatrix} 1.67 & -0.33 & -0.33 \\ -0.33 & 2.17 & -0.83 \\ -0.33 & -0.83 & 2.17 \end{bmatrix} \tag{4.5}$$

We seek the smallest eigenpair, $\lambda_1 = 1$ with eigenvector

$$x_1 = \begin{bmatrix} \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{3}} \end{bmatrix} \tag{4.6}$$

using an initial subspace of

$$x = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{4.7}$$

The colored plane represents the space orthogonal to our subspace. We would like to find the update vector d minimizing the quantity $(x + d)^T A (x + d)$ in that subspace (i.e. $x^T d = 0$) [1]. The light yellow oval denotes the area of the subspace where that

[1]To make the plot more intuitive, we have chosen to refer to our updated vector as $v = x + d$ rather than $v = x - d$.

quantity is smallest, and the dark blue denotes the area where the quantity is large. The increment d which minimizes that trace is

$$d = \begin{bmatrix} 0.2857 \\ 0.4286 \\ 0 \end{bmatrix} \tag{4.8}$$

Note that $x + d$ is much closer to our true solution than our initial guess x. We now turn our attention to how to solve this constrained minimization problem for larger matrices.

We can transform our constrained minimization problem to an unconstrained minimization problem using Lagrange's theorem, which leads to the following saddle point problem for Δ_k

$$\begin{bmatrix} A & BY_k \\ Y_k^T B & 0 \end{bmatrix} \begin{bmatrix} \Delta_k \\ L_k \end{bmatrix} = \begin{bmatrix} AY_k \\ 0 \end{bmatrix} \tag{4.9}$$

where L_k represents the Lagrange multipliers. Alternatively, we may write the above saddle-point problem as

$$\begin{bmatrix} A & BY_k \\ Y_k^T B & 0 \end{bmatrix} \begin{bmatrix} V_{k+1} \\ \bar{L}_k \end{bmatrix} = \begin{bmatrix} 0 \\ I_p \end{bmatrix} \tag{4.10}$$

where $V_{k+1} = Y_k - \Delta_k$ and $\bar{L}_k = -L_k$. Assuming our matrix A is symmetric positive definite, we have satisfied the second order sufficient conditions for optimality; Δ_k is guaranteed to be the solution of our original constrained minimization problem. If A is indefinite, we have no such guarantee, but as our results demonstrate, TraceMin is still capable of computing the smallest eigenpairs. After V_{k+1} is obtained, we B-orthonormalize it and use the Rayleigh-Ritz procedure to generate Y_{k+1}. This process is summarized in Algorithm 1.

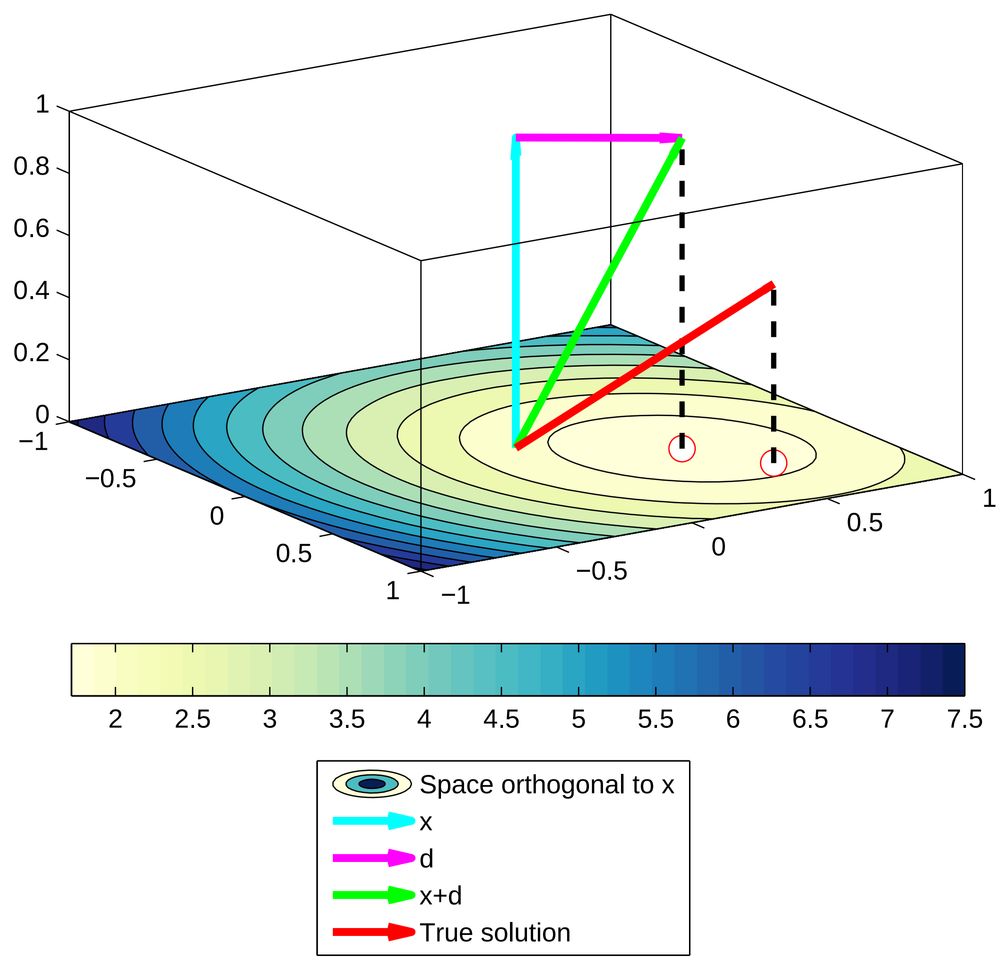


Figure 4.1.: Graphical demonstration of the TraceMin algorithm

Now that the TraceMin algorithm has been presented, we now turn our attention to its convergence properties and some implementation details.

Algorithm 1 TraceMin algorithm

```
Require: Subspace dimension $s > p$,
    $V_1 \in \mathbb{R}^{n \times s}$ with rank $s$,
    $A$ and $B$ symmetric, with $B$ also positive definite
1: for $k = 1 \to$ maxit do
2:     B-orthonormalize $V_k$
3:     Perform the Rayleigh-Ritz procedure to obtain the approximate eigenpairs
       ($AY_k \approx BY_k\Theta_k$):
       • Form $H_k = V_k^T AV_k$
       • Compute all eigenpairs of $H_k$, $H_kX_k = X_k\Theta_k$
       • Compute the Ritz vectors $Y_k = V_kX_k$
4:     Compute the residual vectors $R_k = AY_k - BY_k\Theta_k$
5:     Test for convergence
6:     Solve the saddle point problem (4.10) approximately to obtain $V_{k+1}$
7: end for
```

4.2 Convergence rate

TraceMin is globally convergent, and if $y_{k,i}$ is the ith column of Y_k, the column $y_{k,i}$, converges to the eigenvector x_i corresponding to λ_i for $i = 1, 2, \cdots, p$ with an asymptotic rate of convergence bounded by λ_i/λ_{s+1}, where s is the subspace dimension (or number of vectors in Y). As a result, eigenvalues located closer to the origin will converge considerably faster than ones near λ_{s+1}.

We now turn our attention to a synthetic test matrix which demonstrates this convergence rate in practice. Our synthetic test matrix is order 100, with eigenvalues (0.01, 0.1, 0.5, 0.904, 0.905,...,0.999,1). We will run TraceMin with a subspace dimension of four vectors and examine how long it takes each vector to converge to an absolute residual $\|r = Ay - \theta y\|_2 < 10^{-6}$. Note that the first vector will have a convergence rate of $\frac{0.01}{0.905} \approx 0.011$, and the last vector will have a convergence rate of $\frac{0.904}{0.905} \approx 0.999$. That means TraceMin should take less than ten iterations to compute the first (smallest) eigenpair, but it will take thousands to compute the fourth. Figures 4.2a and 4.2b show the absolute residual $\|r_i = Ay_i - \theta_i y_i\|_2$ and absolute error $e_i = |\theta_i - \lambda_i|$ measured across 50 TraceMin iterations.

In practice, we generally can not measure the error, as we do not know the eigenvalues of interest; we can only measure the residual. It is important to note that while the error decreases monotonically, the residual may not. In this case, the initial Ritz values (the approximate eigenvalues) are in the range (0.9,0.96). These Ritz values are very close to true eigenvalues, but those are not the eigenvalues we seek. The residual only tells us whether a given Ritz pair approximates *some* eigenpair, not whether it approximates the one we want. This is why the residual appears to spike in Figure 4.2a, while the error decreases monotonically in Figure 4.2b.

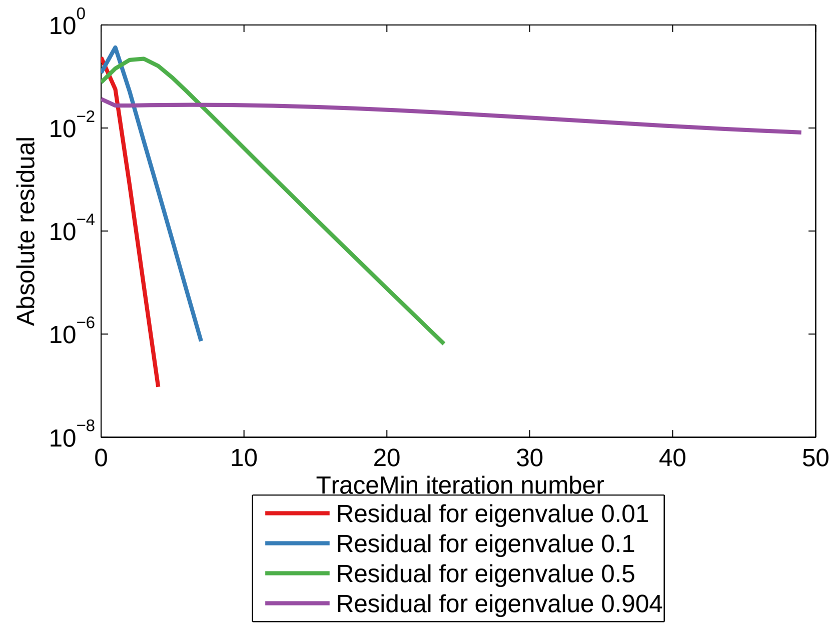
Since we have so far only discussed the subspace dimension as some constant, we now turn our attention to its impact on convergence and how it should be chosen. Later, we will also examine how to improve the convergence rate of TraceMin via shifting.

4.3 Choice of the subspace dimension

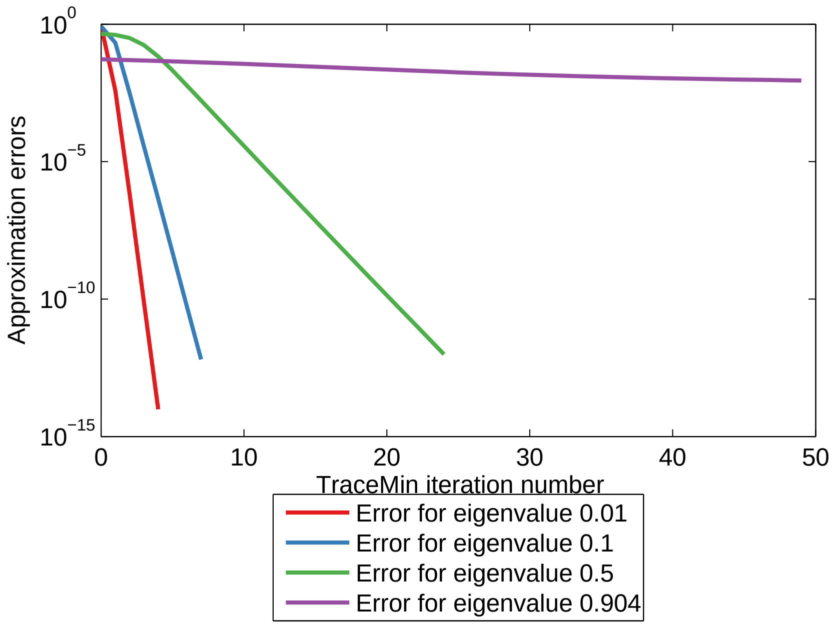
TraceMin uses a constant subspace dimension s, where s is the number of vectors in V. The choice of this subspace dimension s is very important. Larger subspace dimensions may cut down on the number of TraceMin iterations required (because the convergence rate λ_i/λ_{s+1} improves), but each iteration then involves more work. Small subspace dimensions reduce the amount of work done per TraceMin iteration but result in a worse convergence rate. To demonstrate the effect of the subspace dimension on overall work, we now present an example of what happens when we vary the subspace dimension.

This synthetic test matrix is order 100, with eigenvalues

$$(0.1, 0.11, \ldots, 0.16, 0.17, 0.909, 0.91, \ldots, 0.999, 1)$$



(a) Absolute residual for each eigenvalue



(b) Absolute error for each eigenvalue

Figure 4.2.: Demonstration of TraceMin's convergence rate

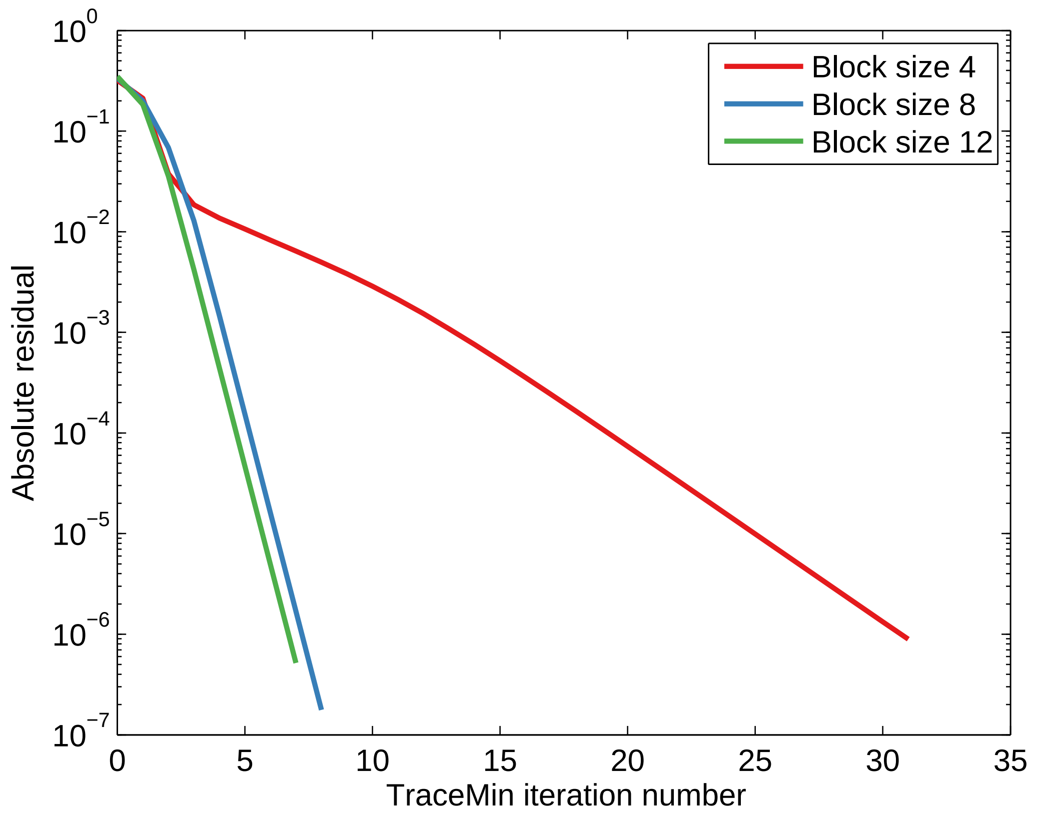


Figure 4.3.: Demonstration of the importance of the block size. This figure presents the residual of the fourth eigenvalue vs number of TraceMin iterations

We will run TraceMin with a subspace dimension of four vectors, then eight vectors, and finally with twelve vectors and examine how many iterations it takes TraceMin to converge. Figure 4.3 shows that increasing the block size for this matrix decreased the number of required TraceMin iterations. Increasing from $s = 4$ to $s = 8$ had a greater impact than increasing from $s = 8$ to $s = 12$ since this problem featured a large gap between the eighth and ninth eigenvalues. For this problem, it's clear that a subspace dimension of $s = 8$ is optimal, given the eigenvalue distribution. We generally can not determine the optimal subspace dimension, since we do not have enough information about the spectrum. In practice, $s = 2p$ tends to work well (where p is the desired number of eigenvalues).

For the tiny examples we have presented so far, the saddle point problems were solved directly. When the matrices become larger, this may be unreasonable. We now explore the effect of using an iterative method to solve the saddle point problem.

4.4 TraceMin as a nested iterative method

In TraceMin, the saddle point problem does not need to be solved to a high degree of accuracy to preserve its global convergence, e.g. see [9], and [17]. Hence, one can use an iterative method with a modest relative residual as a stopping criterion. We will later compare the various saddle point solvers presented in the previous chapter, but for now we concentrate on the selection of the inner (Krylov) tolerance. To demonstrate the impact of the inner tolerance on the convergence of TraceMin, we will study two synthetic examples in which we seek the smallest eigenpair with a *subspace dimension of one vector* [2]. We converge when the relative residual $\|r_i\|_2 / \lambda_i < 10^{-3}$. One of these examples involves a matrix with poorly separated eigenvalues, and the other involves a matrix with well separated eigenvalues.

The first synthetic example is a 100x100 matrix with a condition number of approximately 200. Its two smallest eigenvalues are 4.29e-2 and 4.34e-2. Note that these eigenvalues are clustered, so TraceMin will take many iterations to converge, regardless of how accurately we solve the saddle point problem. First, we will use a direct solver to provide a lower bound on the number of TraceMin iterations required, then we will try projected-CG with various tolerances. In Figure 4.4a, we see that it took TraceMin roughly 180 iterations to converge, regardless of whether we used a direct solve or an iterative method with a moderate tolerance. If we require a modest residual in the linear solve (a tolerance of 0.5), it only takes 20 more TraceMin iterations than if we had used a direct solve. Figure 4.4b shows that the overall work required by TraceMin with an inaccurate Krylov solver is far less than with the stricter tolerances. This example demonstrates that it does not matter how

[2]In general, it is a poor decision to use such a small subspace dimension.

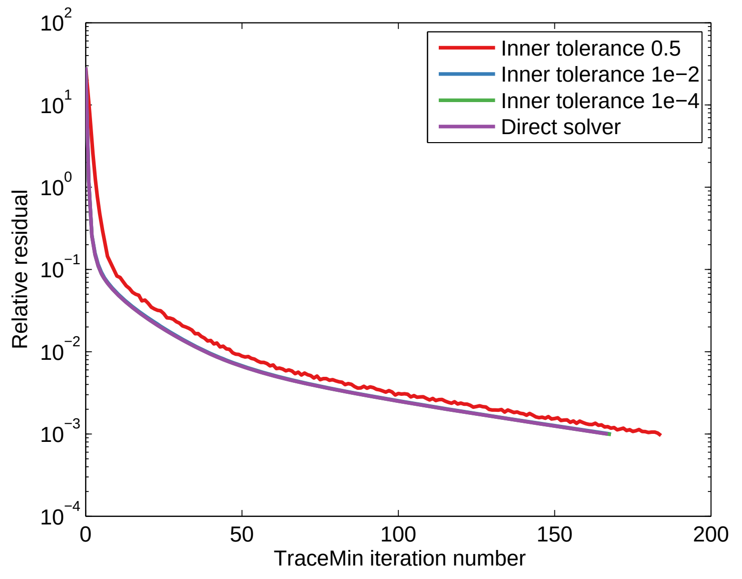
accurately we solve the saddle point problem if TraceMin's convergence rate is poor due to clustered eigenvalues and too small a block size s.

The second synthetic example is a 100x100 matrix with a condition number of approximately 2e7. Its two smallest eigenvalues are 3.58e-7 and 3.58e-3. Note that these are well separated eigenvalues, so TraceMin, with a block size $s = 1$, will converge in a small number of iterations if a direct solver is used. Unlike the previous example, Figure 4.5a shows there is a dramatic difference in the number of TraceMin iterations based on the inner projected-CG tolerance. If we use a relatively strict tolerance of 1e-4, TraceMin converges in only four iterations; with a tolerance of 0.5, TraceMin takes 25 iterations to converge. When we examine the number of projected-CG iterations in Figure 4.5b, we see that in this case, it was more efficient to use a stricter tolerance because of the separation of eigenvalues.

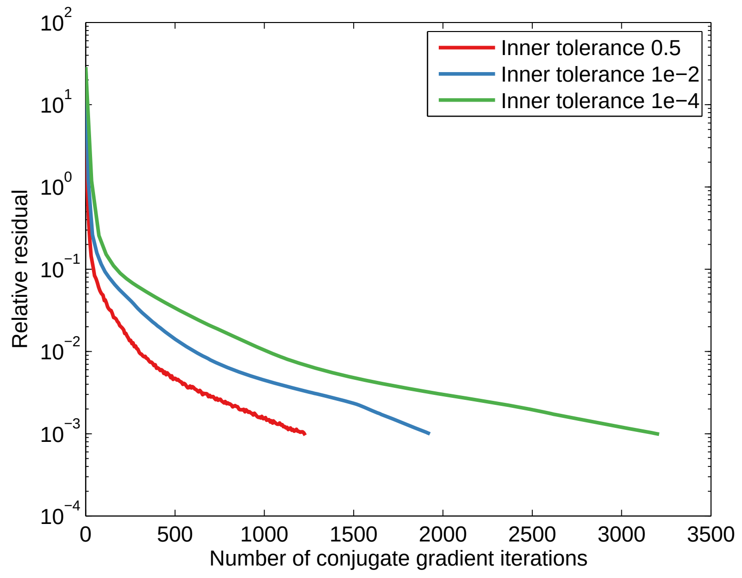
The moral is, the more clustered the eigenvalues are, the less important it is to solve the linear systems accurately. However, we generally know very little about the clustering of the eigenvalues prior to running TraceMin. We can attempt to estimate the convergence rate of each eigenpair by using the Ritz values, but the Ritz values tend to be very poor estimates of the eigenvalues for at least the first few TraceMin iterations. In our TraceMin implementation, We compensate for this by choosing the tolerance based on both the Ritz values and the current TraceMin iteration. The exact equation we use is as follows

$$tol_i = \min\left(\frac{\theta_i}{\theta_s}, 2^{-j}\right) \tag{4.11}$$

where i is the index of the targetted right hand side, j is the current TraceMin iteration number, and θ are the current Ritz values. Since this expression does not make sense for $i = s$, we choose $tol_s = tol_{s-1}$. We also specify a maximum number of Krylov iterations to be performed, since TraceMin does not rely on accurate linear solves to converge.

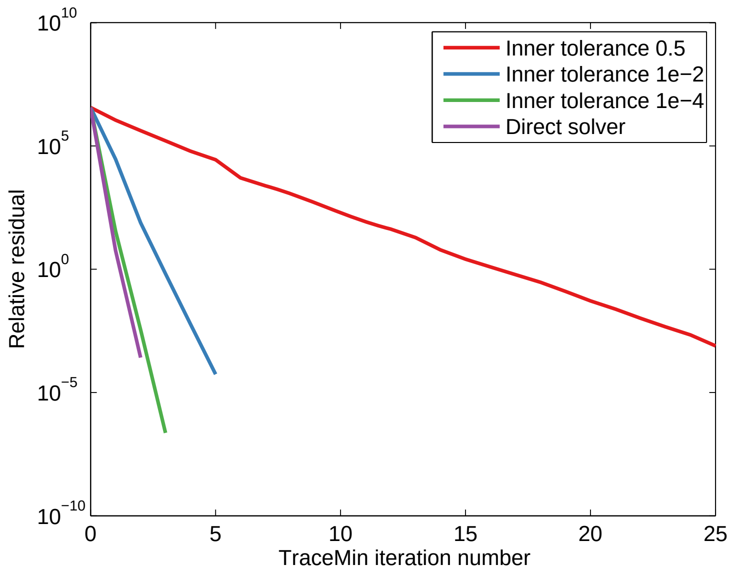


(a) Residual of the smallest eigenvalue vs number of TraceMin iterations

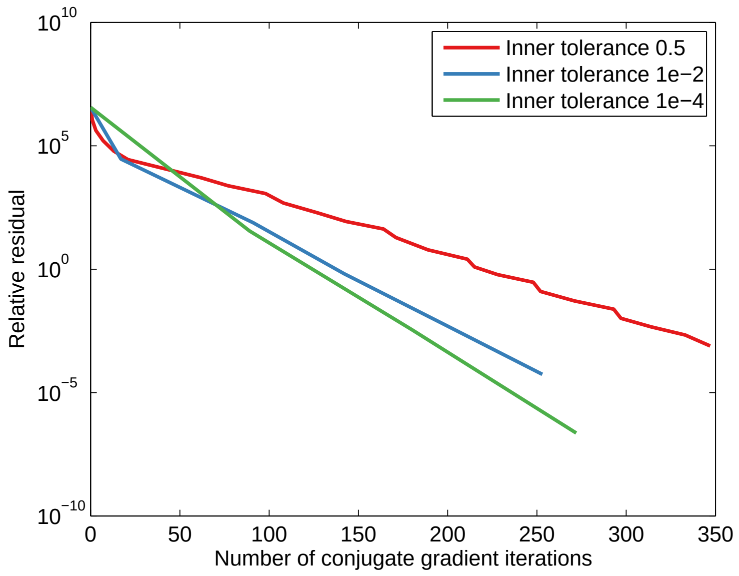


(b) Residual of the smallest eigenvalue vs number of projected-CG iterations

Figure 4.4.: Demonstration of the importance of the inner Krylov tolerance for a problem with poorly separated eigenvalues



(a) Residual of the smallest eigenvalue vs number of TraceMin iterations



(b) Residual of the smallest eigenvalue vs number of projected-CG iterations

Figure 4.5.: Demonstration of the importance of the inner Krylov tolerance for a problem with well separated eigenvalues

4.5 Deflation of converged eigenvectors

So far, we have not discussed what to do when an eigenpair converges. We would like to remove it from our subspace V so that we do not continue to do unnecessary work improving a vector which has already converged. However, we need to ensure that after we remove a converged vector from the subspace, the subspace stays B-orthogonal to it, or else we will converge to the same vector over and over again. If C is our set of converged eigenvectors, the projector

$$P = I - BC \left(C^T B^2 C\right)^{-1} C^T B \tag{4.12}$$

applied to our subspace V will preserve that condition by forcing $PV \perp_B C$. This process of projecting the converged vectors from the subspace is called *deflation*[3]. If we add this feature to Algorithm 1, we end up with Algorithm 2. The few steps this adds to the TraceMin iterations have been highlighted in red.

After a Ritz vector converges, we may either remove it from the subspace and continue to work with a smaller subspace of dimension $s - 1$, or we may replace it with a random vector. If we do not replace the converged vector, our linear systems have one fewer right hand side, and TraceMin will require less work per iteration. However, if we replace the converged vector with a random one, the convergence rate for the nonconverged Ritz vectors will improve and we will require fewer TraceMin iterations overall. In our implementation, we replace the converged vectors with random ones and hold the subspace dimension constant.

4.6 Ritz shifts

The convergence rate of TraceMin is based on the location of the eigenvalues of interest within the spectrum. As we have seen, if they are far from the origin, the rate of convergence is very poor. Therefore, it can be worthwhile to perform a shift which

[3]The Trilinos documentation refers to this as *locking*, but it is the same concept.

Algorithm 2 TraceMin algorithm (with deflation)

```
Require: Subspace dimension $s > p$,
    $V_1 \in \mathbb{R}^{n \times s}$ with rank $s$,
    $A$ and $B$ symmetric, with $B$ also positive definite
 1: for $k = 1 \to$ maxit do
 2:   if $C$ is not empty then
 3:     Perform the projection $V_k = PV_k$, where $P = I - BC\left(C^T B^2 C\right)^{-1} C^T B$
 4:   end if
 5:   B-orthonormalize $V_k$
 6:   Perform the Rayleigh-Ritz procedure to obtain the approximate eigenpairs
      $(AY_k \approx BY_k\Theta_k)$
 7:     Form $H_k = V_k^T A V_k$
 8:     Compute all eigenpairs of $H_k$, $H_k X_k = X_k \Theta_k$
 9:     Compute the Ritz vectors $Y_k = V_k X_k$
10:   Compute the residual vectors $R_k = AY_k - BY_k\Theta_k$
11:   Test for convergence
12:   Move converged vectors from $Y$ to $C$
13:   Solve the following saddle point problem approximately to obtain $V_{k+1}$

      $$\begin{bmatrix} A & BY_k & BC \\ Y_k^T B & 0 & 0 \\ C^T B & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta_k \\ L_k^{(1)} \\ L_k^{(2)} \end{bmatrix} = \begin{bmatrix} AY_k \\ 0 \\ 0 \end{bmatrix} \tag{4.13}$$

14: end for
```

moves the desired eigenvalues closer to the origin. Instead of solving our original problem $Ax = \lambda Bx$, we solve the problem $(A - \omega B)\,x = (\lambda - \omega)\,Bx$, where ω is our shift. The convergence rate for eigenpair i is now

$$\frac{\lambda_i - \omega}{\lambda_{s+1} - \omega} \tag{4.14}$$

rather than λ_i/λ_{s+1}. If $\omega \approx \lambda_i$, eigenpair i will converge very quickly. The only change these shifts necessitate in the TraceMin algorithm is that the saddle point problem of Equation 4.9 becomes

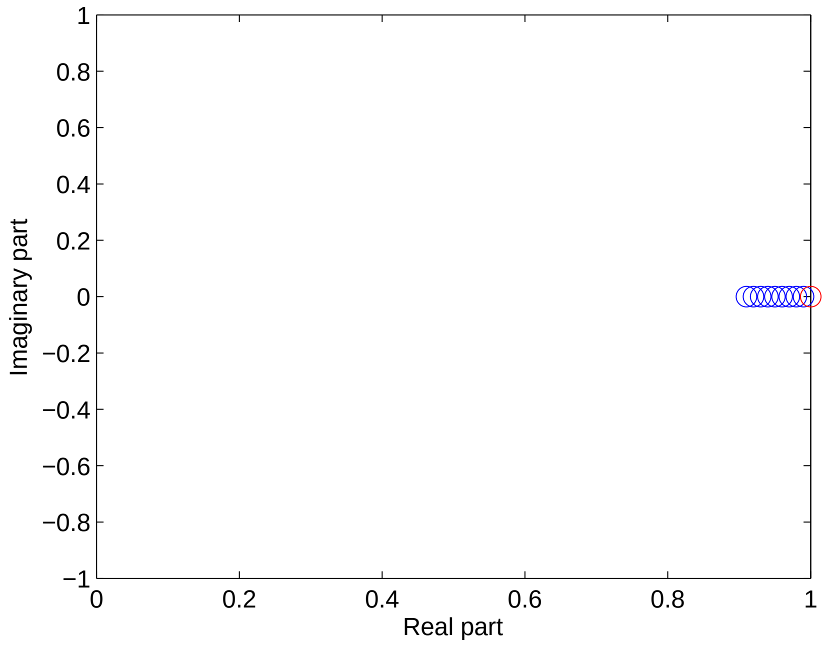
$$\begin{bmatrix} (A - \omega B) & BY_k \\ BY_k^T & 0 \end{bmatrix} \begin{bmatrix} \Delta_{k+1} \\ L_k \end{bmatrix} = \begin{bmatrix} (A - \omega B)\,Y_k \\ 0 \end{bmatrix} \tag{4.15}$$

The matrix $A - \omega B$ may be formed explicitly, or it may be applied implicitly [4]

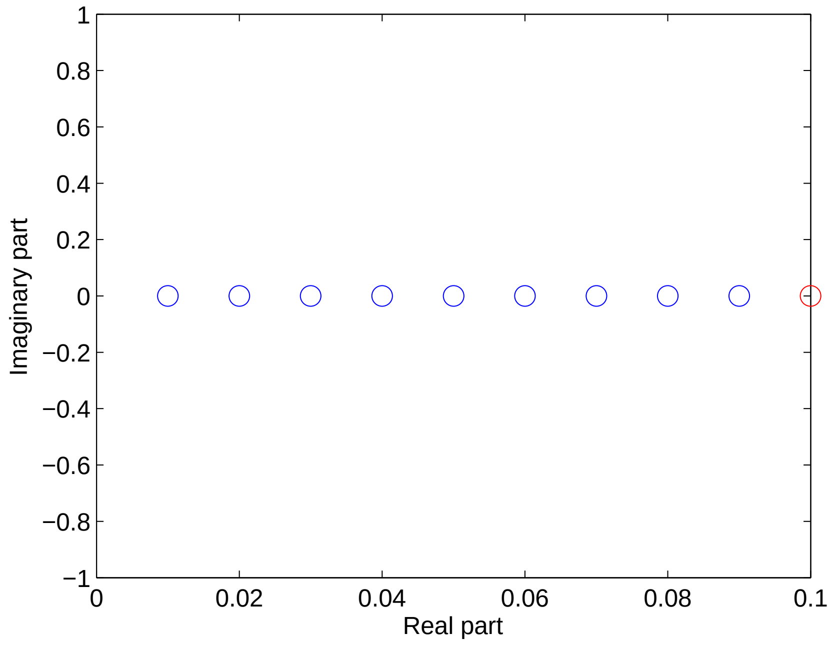
We now present a small synthetic test problem demonstrating the effect of these shifts. Suppose we wish to find the four smallest eigenpairs of a test matrix with an absolute residual of 10^{-5}. This test matrix has 1000 rows, and its eigenvalues lie evenly spaced in the interval $[0.91, 10.9]$. We will run TraceMin twice using a subspace dimension of nine vectors. The first time, we will use the original matrix without a shift, and then we will try TraceMin with a shift of 0.9. Note that 0.9 is a close approximatiion of the smallest eigenvalue.

The original matrix has an unfavorable eigenvalue distribution (Figure 4.6); the eigenvalues we seek are very far from the origin and close to $\lambda_{10} = 1$, i.e. the convergence rate is practically 1. The shifted matrix exhibits a much better eigenvalue distribution. Some of the eigenvalues are still very far from the origin, but the four targetted eigenpairs are much closer. We see from figure 4.7 that it takes roughly 180 iterations of TraceMin to solve the problem without shifting, but it takes only 12 iterations to solve the problem with the shift because we improved the eigenvalue distribution.

[4]In our Trilinos implementation, $A - \omega B$ is applied implicitly. We do this to accomodate for the case where A and B are not available explicitly.

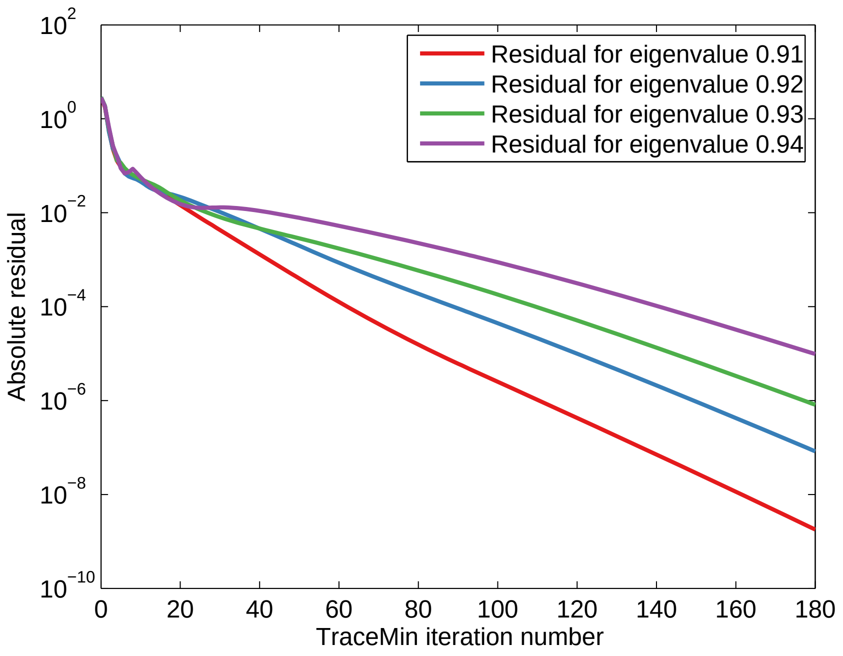


(a) Original eigenvalue distribution

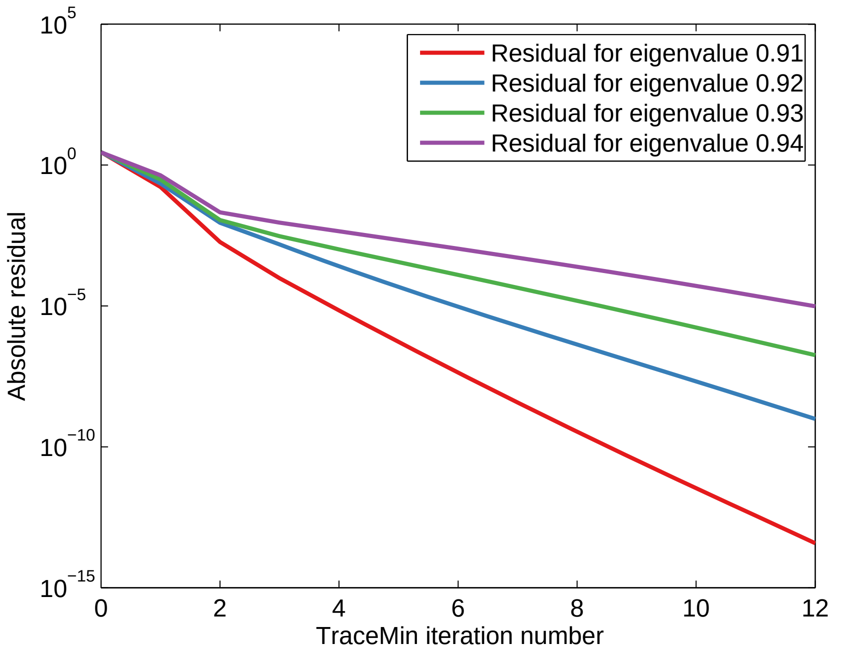


(b) Shifted eigenvalue distribution

Figure 4.6.: The effect of Ritz shifts on the eigenvalue spectrum



(a) Original convergence rate



(b) Improved convergence rate

Figure 4.7.: The effect of Ritz shifts on convergence

4.6.1 Multiple Ritz shifts

In the previous example, we used a single shift for all of the Ritz pairs. This improved the convergence rate of all eigenpairs, but it had the greatest effect on the smallest one (since the shift so closely approximated the smallest eigenvalue). Instead of using a single shift, we could use separate shifts for each of the Ritz pairs. That would result in solving s saddle point problems per TraceMin iteration of the form[5]

$$\begin{bmatrix} (A - \omega_i B) & BY \\ BY^T & 0 \end{bmatrix} \begin{bmatrix} d_i \\ l_i \end{bmatrix} = \begin{bmatrix} (A - \omega_i B)\, y_i \\ 0 \end{bmatrix} \tag{4.16}$$

Note that these saddle point problems do not need to be solved separately. We may use a pseudo-block Krylov method to solve these linear systems, but not a block Krylov method[6].

If each shift closely approximates the corresponding eigenvalue, the convergence rate of *every* eigenpair would be greatly improved, rather than just the convergence rate of the smallest. In the following example, we see how the use of multiple shifts impacts the convergence rate of TraceMin.

A is a synthetic test matrix of order $n = 100$ whose eigenvalues lie evenly spaced in the interval $[0.91, 10.9]$. We are looking for the four smallest eigenpairs using a subspace of dimension 9, and we want an absolute residual of 1e-5. We will try TraceMin with no shifts, with a single shift of 0.9, and with multiple shifts (0.9, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7). Note that for each Ritz shift ω_i, $\omega_i \approx \lambda_i$. Figure 4.8 shows that without shifts, TraceMin will not converge very quickly for this problem. If we use a single shift of 0.9, the smallest eigenvalue will converge quickly, but the others will take much longer. If we use multiple shifts, each of the desired eigenvalues should converge in only a few iterations.

[5] We have dropped the TraceMin iteration subscript k for clarity.

[6] Pseudo-block Krylov methods are mathematically equivalent to solving each linear system independently. The only difference is that in an MPI program, several messages may be grouped together, resulting in a lower communication cost. Block Krylov methods build one subspace which is used

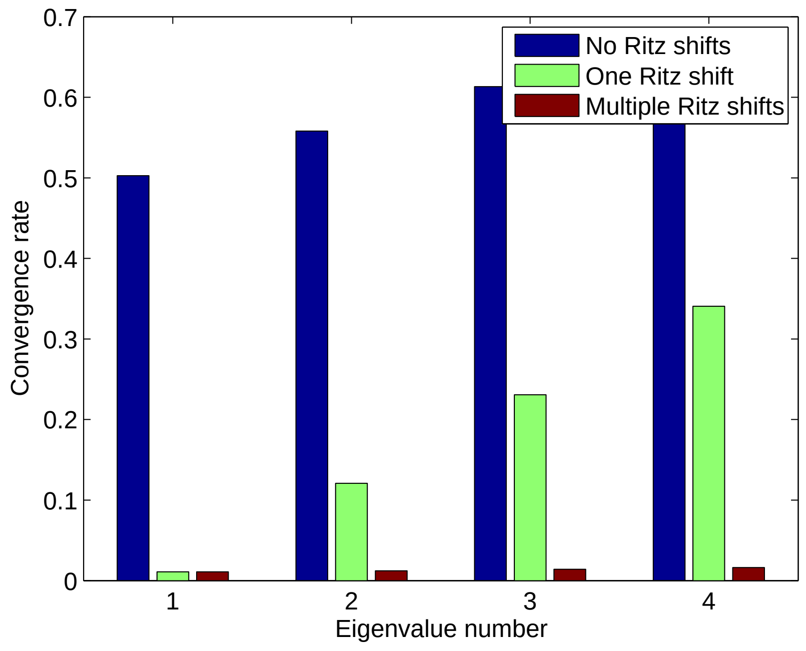


Figure 4.8.: The effect of multiple Ritz shifts on TraceMin's convergence rate

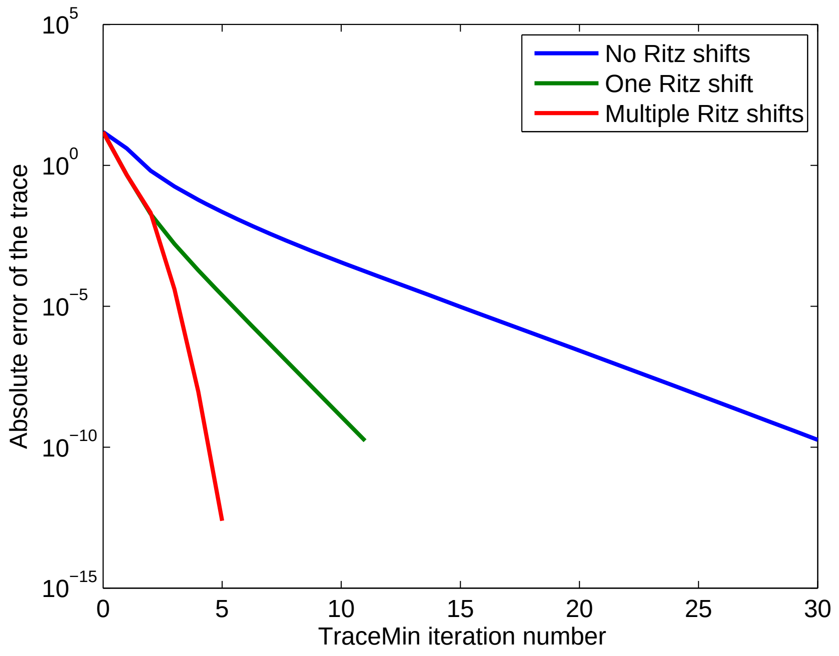


Figure 4.9.: The effect of multiple Ritz shifts on the trace reduction

Figure 4.9 shows us that the use of multiple shifts reduces the trace of $Y^T AY$ much faster than using only one shift, or no shifts at all. Figure 4.10 shows that the use of multiple shifts also lowers the residual of each of the four smallest Ritz pairs much faster than the other shifting strategies. With the multiple shifts, it took only five iterations for TraceMin to find the four smallest eigenpairs. Using a single shift resulted in convergence after eleven TraceMin iterations. Without shifts, TraceMin required 30 iterations to converge.

We now consider how to choose the optimal shift based on nothing but the Ritz values (approximate eigenvalues) and their corresponding residuals.

for all right hand sides, rather than handling each independently. It would not make sense to use a block Krylov method with these saddle point problems.

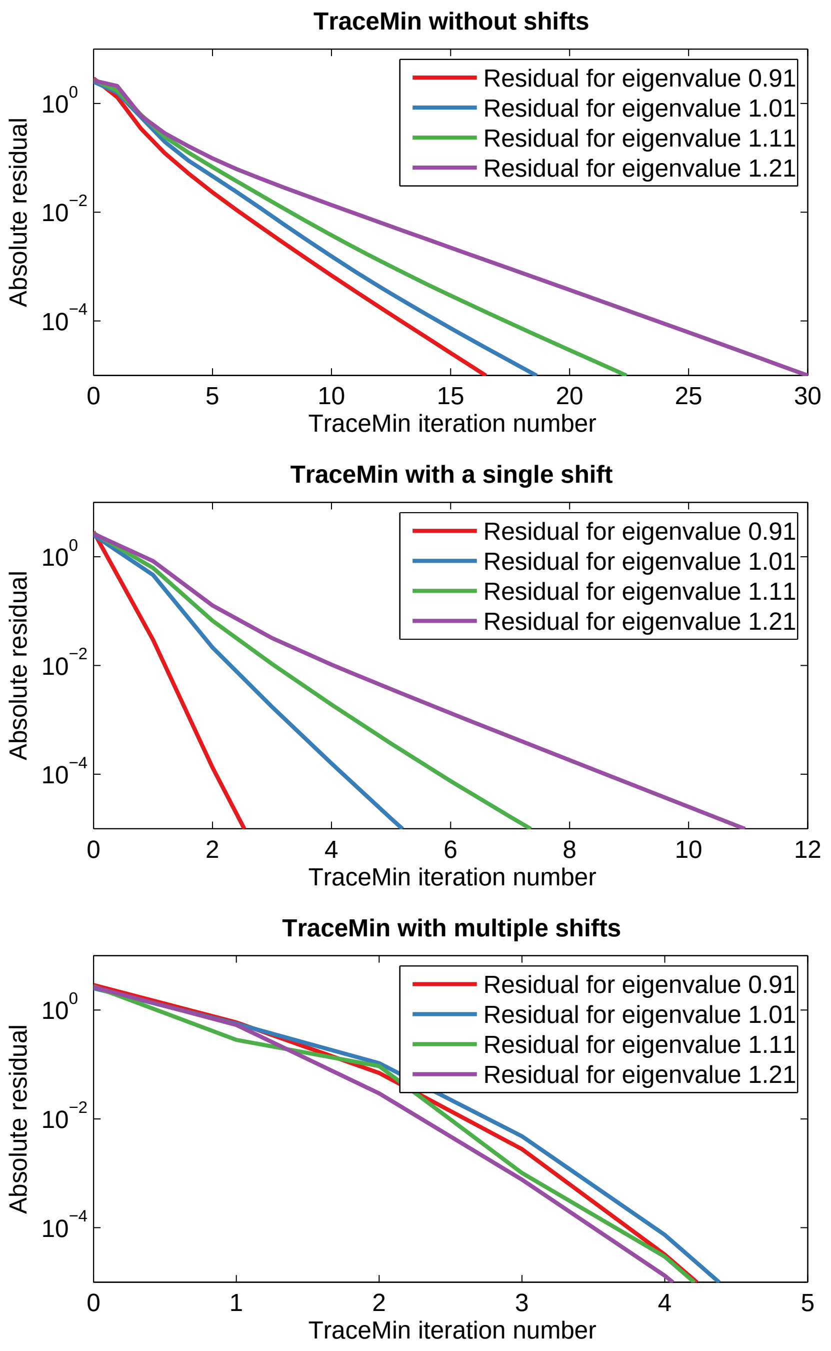


Figure 4.10.: The effect of multiple Ritz shifts on convergence

4.6.2 Choice of the Ritz shifts

Choosing how and when to shift is a difficult issue. If we shift too aggressively, we run the risk of converging to a completely different set of eigenpairs than the ones we seek, and global convergence is destroyed. Shifting very conservatively avoids that problem, but it is detrimental to the overall running time of the program because we performed many unnecessary TraceMin iterations. In our TraceMin implementation, We allow the user to choose just how aggressive he wishes to be with the shifts, although the default options tend to work well. The user can choose to shift at every iteration, after the trace has leveled (i.e. when the relative change in trace between successive iterations has become smaller than a user defined tolerance), or he can choose to disable shifting entirely. The user may choose the shifts as being equal to the largest converged eigenvalue, the adjusted Ritz values (which are essentially computed as $\theta_i - \|r_i\|_2$ and are described in Algorithm 3), or the current Ritz values. He may also choose whether to use a single Ritz shift or separate ones for each Ritz pair. Our default method of shifting is presented in Algorithm 3, which is largely based on the work of [17].

4.7 Relationship between TraceMin and simultaneous iteration

The method of simultaneous iteration (Algorithm 4) was developed by Friedrich Bauer in 1957 under the name *Treppeniteration* [18]. Note that this method is mathematically equivalent to TraceMin, if we solve the saddle point problem by computing the Schur complement [7]. The difference between the original 1982 TraceMin algorithm and simultaneous iteration is that TraceMin (using a projected Krylov method to solve the saddle point problem) enforces a condition that Δ_k, the update to Y_k, must be B-orthogonal to the current Ritz vectors Y_k, whereas simultaneous iteration only enforces that condition if the linear systems $AV = BY$ are solved to a high

[7]If $Z_k \approx A^{-1}BY_k$, TraceMin computes $V_{k+1}^{(t)} = Z_k \left(Y_k^T B Z_k\right)^{-1}$, whereas the method of simultaneous iteration computes $V_{k+1}^{(s)} = Z_k$. $V_{k+1}^{(t)}$ and $V_{k+1}^{(s)}$ span the same subspace.

Algorithm 3 Default shift-selection algorithm

```
Require: Subspace dimension $s$
    Computed residual $R = AY - BY\Theta$
 1: Determine whether the Ritz values are clustered.
    Ritz values $\theta_i$ and $\theta_{i+1}$ are in a cluster if $\theta_i + \|r_i\|_2 \geq \theta_{i+1} - \|r_{i+1}\|_2$
 2: For each cluster, compute the residual norm of that cluster.
    If $\theta_i$, $\theta_j$, and $\theta_k$ are in a cluster, the residual norm of that cluster is $\beta_{i,j,k} = \|r_i\|_2 + \|r_j\|_2 + \|r_k\|_2$
 3: if at least one eigenvalue has converged then
 4:   $\omega_1$ = the largest converged eigenvalue
 5: else
 6:   $\omega_1 = 0$
 7: end if
 8: if $\theta_1$ is not in a cluster with $\theta_2$ then
 9:   $\omega_1 = \max(\omega_1, \theta_1)$
10: else
11:   $\omega_1 = \max(\omega_1, \theta_1 - \beta_1)$
12: end if
13: for $k = 2 \to s - 1$ do
14:   if $\omega_{k-1} = \theta_{k-1}$ and $\theta_k$ is not in a cluster with $\theta_{k+1}$ then
15:     $\omega_k = \theta_k$
16:   else if there exists a $\theta_i$ such that $\theta_i < \theta_k - \|r_k\|_2$ then
17:     $\omega_k$ = the largest $\theta_i$ satisfying that condition
18:   else
19:     $\omega_k = \omega_{k-1}$
20:   end if
21: end for
22: $\omega_s = \omega_{s-1}$
```

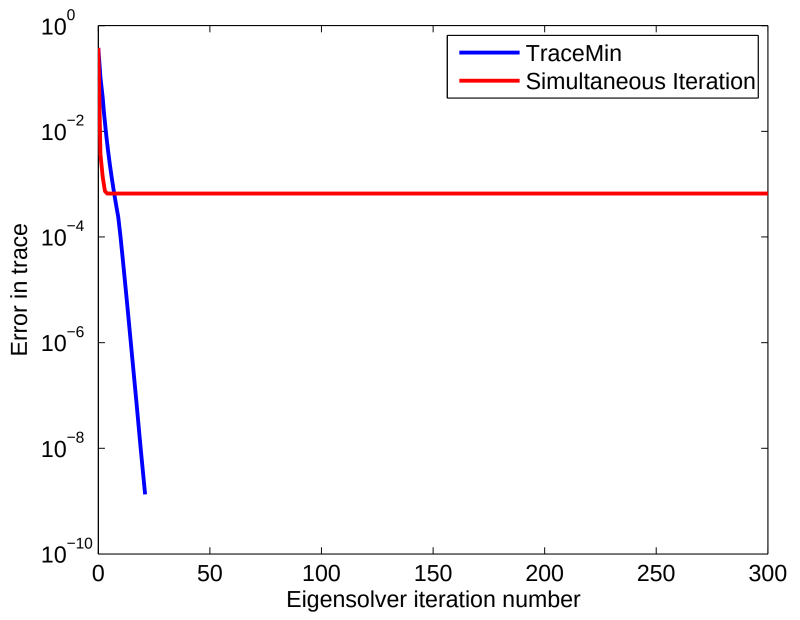
degree of precision. As a result, TraceMin tends to perform better than simultaneous iteration when the linear systems are solved inexactly. We now look at an example comparing the two methods.

Algorithm 4 Simultaneous iteration algorithm

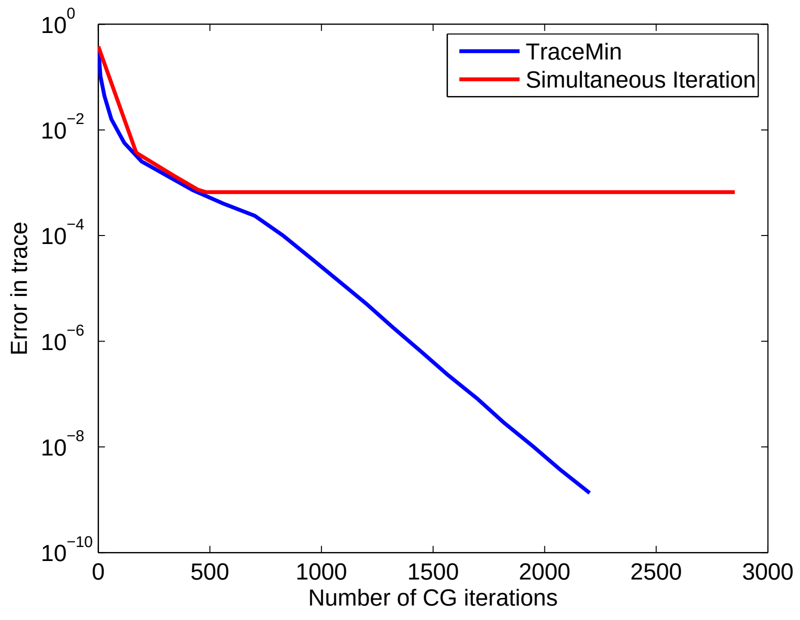
```
Require: Subspace dimension $s > p$,
    $V_1 \in \mathbb{R}^{n \times s}$ with rank $s$,
    $A$ and $B$ symmetric, with $B$ also positive definite
 1: for $k = 1 \to$ maxit do
 2:     B-orthonormalize $V_k$
 3:     Perform the Rayleigh-Ritz procedure to obtain the approximate eigenpairs
        ($AY_k \approx BY_k\Theta_k$)
 4:         Form $H_k = V_k^T A V_k$
 5:         Compute all eigenpairs of $H_k$, $H_k X_k = X_k \Theta_k$
 6:         Compute the Ritz vectors $Y_k = V_k X_k$
 7:     Compute the residual vectors $R_k = AY_k - BY_k\Theta_k$
 8:     Test for convergence
 9:     Solve the set of linear systems $AV_{k+1} = BY_k$
10: end for
```

Our test matrix A is a synthetic test matrix of order 100, with a condition number of 1000. We seek the four smallest eigenpairs with an absolute residual of 1e-6, using a subspace dimension of eight. We will try TraceMin, using projected-CG to solve the saddle point problem, and simultaneous iteration, with CG to solve $AV_{k+1} = BY_k$. Figure 4.11 shows what happens if we use a very modest tolerance of 0.5 when solving the linear systems arising in each iteration. TraceMin converges in roughly 30 iterations, whereas simultanous iteration fails to converge within 300 iterations. Clearly it was a bad idea to use such a large inner tolerance with simultaneous iteration, so we will try it again with a stricter inner tolerance of 1e-3. Figure 4.12 shows that even with that stricter inner tolerance, simultaneous iteration still failed to converge within 300 iterations. Figure 4.13 presents a comparison of TraceMin and simultaneous iteration with an inner tolerance of 1e-6. Since we solved the linear systems resulting at each iteration with considerably more accuracy, TraceMin and simultaneous iteration converge in the same number of iterations. We also see that they both required roughly the same number of conjugate gradient iterations, which means simultaneous iteration is the faster algorithm in this case[8]. However, if we examine the total number of CG iterations required, the overall best approach to solving this problem was to use TraceMin with an inner tolerance of 0.5, since that required the fewest conjugate gradient iterations overall.

[8]Each conjugate gradient iteration for simultaneous iteration involved applying the operator A to a vector. For TraceMin, each conjugate gradient iteration involved applying the operator PAP, which requires at least one projection

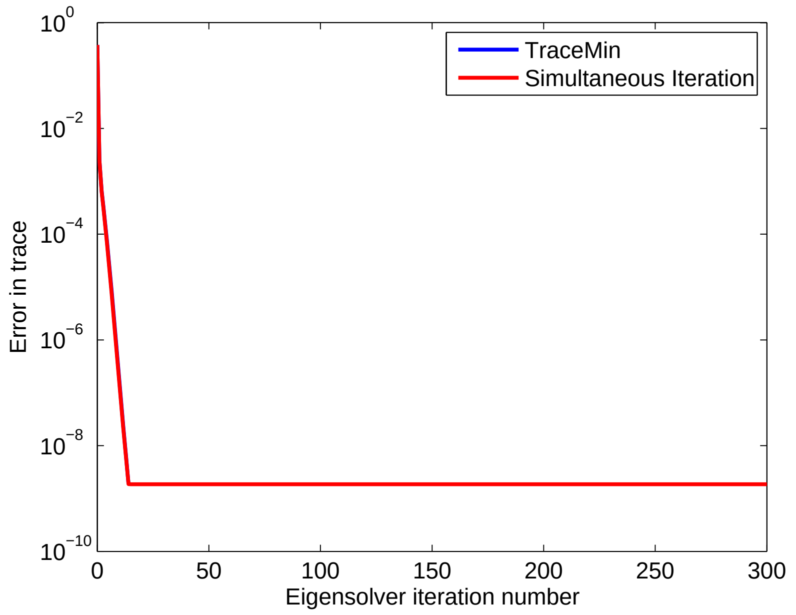


(a) Error in trace vs. number of TraceMin/simultaneous iteration iterations

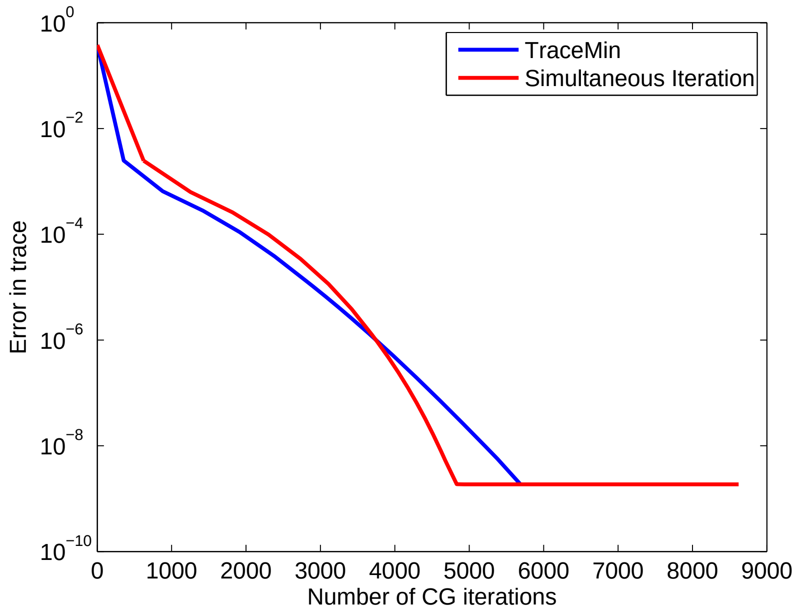


(b) Error in trace vs. number of conjugate gradient iterations

Figure 4.11.: A comparison of TraceMin and simultaneous iterations using a lenient inner tolerance

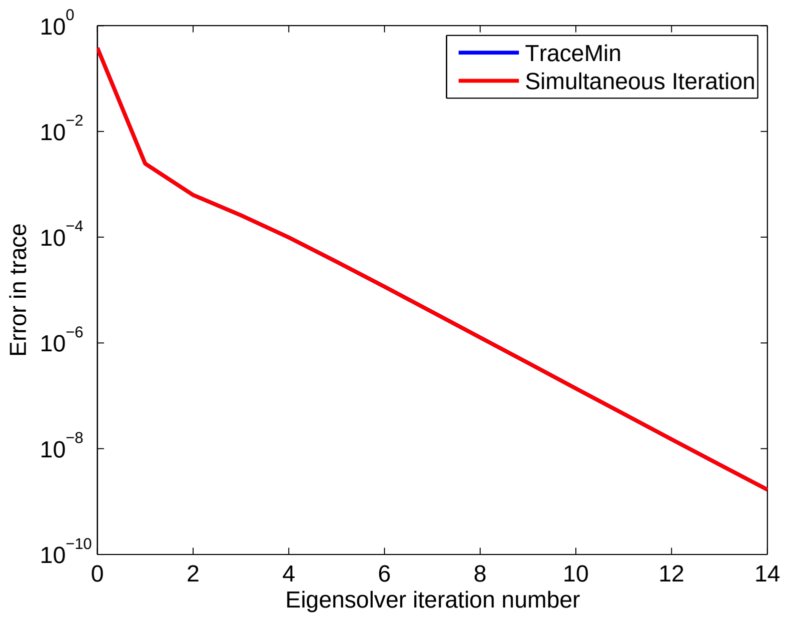


(a) Error in trace vs. number of TraceMin/simultaneous iteration iterations

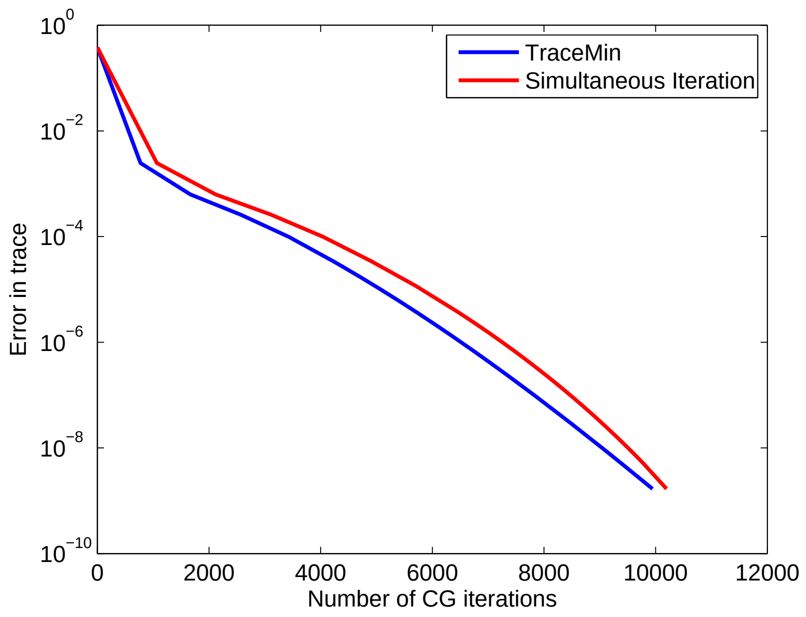


(b) Error in trace vs. number of conjugate gradient iterations

Figure 4.12.: A comparison of TraceMin and simultaneous iterations using a moderate inner tolerance



(a) Error in trace vs. number of TraceMin/simultaneous iteration iterations



(b) Error in trace vs. number of conjugate gradient iterations

Figure 4.13.: A comparison of TraceMin and simultaneous iterations using a strict inner tolerance

5 TRACEMIN-DAVIDSON

TraceMin-Davidson is an eigensolver very similar to TraceMin. The only real difference is that while TraceMin uses a constant subspace dimension, TraceMin-Davidson uses expanding subspaces. In every iteration, we add a set number of vectors to our subspace V. When V gets to be too large, we shrink it, keeping only the most important part of the subspace, i.e. the Ritz vectors corresponding to the smallest Ritz values. Essentially, TraceMin-Davidson is to TraceMin as block-Lanczos is to simultaneous iteration. This difference is outlined in Algorithm 5.

Most of the items explored in the previous chapter still apply here. TraceMin-Davidson converges faster if the eigenvalues are well separated, and we can still use Ritz shifts to improve the convergence rate.

We now explore some of TraceMin-Davidson's implementation issues.

5.1 Minimizing redundant computations

At each TraceMin-Davidson iteration, we add s new vectors to our subspace V, but the rest of the subspace remains constant. In this section, we explore how to take advantage of that fact in order to minimize the amount of required computations.

The B-orthonormalization of V_k can be simplified as follows[1]:

- Project the vectors of BV_{k-1} out of Δ_{k-1}:
  $\Delta_{k-1} \leftarrow \left(I - BV_{k-1}\left(V_{k-1}^T B^2 V_{k-1}\right)^{-1} V_{k-1}^T B\right)\Delta_{k-1}$
- B-orthonormalize Δ_{k-1}

[1]This simplification comes from the fact that V_{k-1} was already B-orthonormalized in the previous iteration.

Algorithm 5 TraceMin-Davidson algorithm

```
Require: Block size $s$
    Maximum subspace dimension $d > 2s$,
    $V_1 \in \mathbb{R}^{n \times s}$ with rank $s$,
    $A$ and $B$ symmetric, with $B$ also positive definite
 1: Initialize current subspace dimension $c = s$
 2: for $k = 1 \to$ maxit do
 3:   B-orthonormalize $V_k$
 4:   Perform the Rayleigh-Ritz procedure to obtain the approximate eigenpairs
      $(AY_k \approx BY_k\Theta_k)$
 5:      Form $H_k = V_k^T AV_k$
 6:      Compute all eigenpairs of $H_k$, $H_kX_k = X_k\Theta_k$
         Assume the eigenvalues are sorted in ascending order $\theta_1 \leq \theta_2 \leq \cdots \leq \theta_c$
 7:      Compute the Ritz vectors $Y_k = V_kX_k$
         Let $Y_{k,s}$ denote the $s$ Ritz vectors corresponding to the smallest Ritz values
 8:   Compute the residual vectors $R_k = AY_k - BY_k\Theta_k$
 9:   Test for convergence
10:   if $c + s > d$ then
11:      Restart with $V_k = Y_{k,s}$ and $c = s$
12:   end if
13:   Solve the saddle point problem
```

$$\begin{bmatrix} A & BY_{k,s} \\ Y_{k,s}^T B & 0 \end{bmatrix} \begin{bmatrix} \Delta_k \\ L_k \end{bmatrix} = \begin{bmatrix} AY_{k,s} \\ 0 \end{bmatrix} \tag{5.1}$$

```
      approximately to obtain $\Delta_k$
14:   Add $\Delta_k$ to the subspace, $V_{k+1} = [V_k \ \Delta_k]$
15: end for
```

- Add Δ_{k-1} to the subspace:
  $V_k = [V_{k-1}\ \Delta_{k-1}]$

Additionally, we only need to compute the s new vectors of AV_k and the corresponding columns of $H_k = V_k^T AV_k$.

5.2 Selecting the block size

Unlike TraceMin, we can run TraceMin-Davidson with a block size smaller than the number of desired eigenpairs. Using a larger block size involves more work per TraceMin-Davidson iteration but has the potential to reduce the number of required TraceMin-Davidson iterations. Additionally, using a block size $s > 1$ allows us to use block operations in the linear solve step, which can be more effective on parallel architectures. One further consideration in choosing the block size is its effect on global convergence. If the block size is smaller than the multiplicity of the eigenvalues sought, we may miss the correct multiplicity. We now demonstrate this with an example.

We seek the four smallest eigenpairs of the 3D discretization of the Laplace operator on a unit cube of order $n = 1000$. The four smallest eigenvalues are approximately $(0.243, 0.480, 0.480, 0.480)$. Note that the second eigenvalue has a multiplicity of three. We will run TraceMin-Davidson twice, once with a block size of $s = 1$, and once with a block size of $s = 4$. Our initial subspace contains four vectors, and we do not use restarts. In both cases, we consider an eigenpair as having converged if the absolute residual $\|r = Ay - \theta By\|_2 < 10^{-5}$. Figure 5.1 shows that with a block size of $s = 4$, TraceMin-Davidson converges to the true eigenvalues (plotted as black circles) after 10 iterations. With a block size of $s = 1$, TraceMin-Davidson reports convergence after 22 iterations. With the larger block size, TraceMin-Davidson would likely scale better due to its capacity to use block operations during the linear solve step. The most important reason to use a larger block size here though is this: when we used a block size of $s = 1$, we did not converge to the correct eigenpairs. TraceMin-Davidson returned four eigenpairs with a residual $\|r\|_2 < 10^{-5}$, but the fourth one it returned was 0.716, which is the fifth eigenvalue of this particular problem. If this were not a small synthetic problem, we would have no idea that the eigenpairs returned by TraceMin-Davidson were incorrect. For this reason, we set the default block size to be the same as the desired number of eigenpairs in our TraceMin-Davidson implementation. If the user has some knowledge about the spectrum, he may choose to use a smaller block size.

5.3 Computing harmonic Ritz values

Each TraceMin-Davidson iteration essentially consists of two parts: compute a B-orthonormal basis V of a subspace $\mathbb{K}$, then compute an approximate eigenvector[2] y based on that subspace such that

$$y \in \mathbb{K} \tag{5.2}$$

and the Galerkin condition

$$r \perp \mathbb{K} \tag{5.3}$$

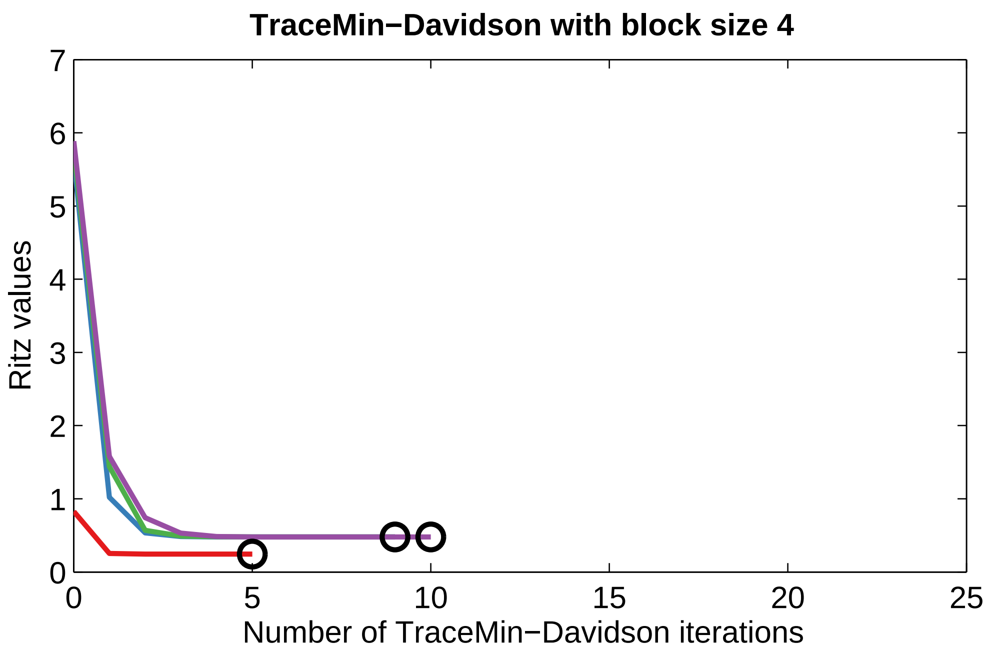
is satisfied, where $r = Ay - \theta By$. From conditions 5.2 and 5.3, we know that $y = Vu$ where u is the eigenvector corresponding to the desired eigenvalue of the following small dense problem

$$V^T AVu = \theta u \tag{5.4}$$

This is commonly known as the Rayleigh-Ritz procedure. These Ritz pairs (θ, y) tend to approximate extreme eigenpairs better than interior ones [19]. If we wish to compute interior eigenpairs, we may instead compute the harmonic Ritz pairs.

Instead of using an orthogonal projection method as we did before, we can use an oblique projection method. Let $\mathbb{K}$ be the space spanned by the vectors of V as before

[2]Here, we focus on the single vector case, but this also applies to blocks of vectors.



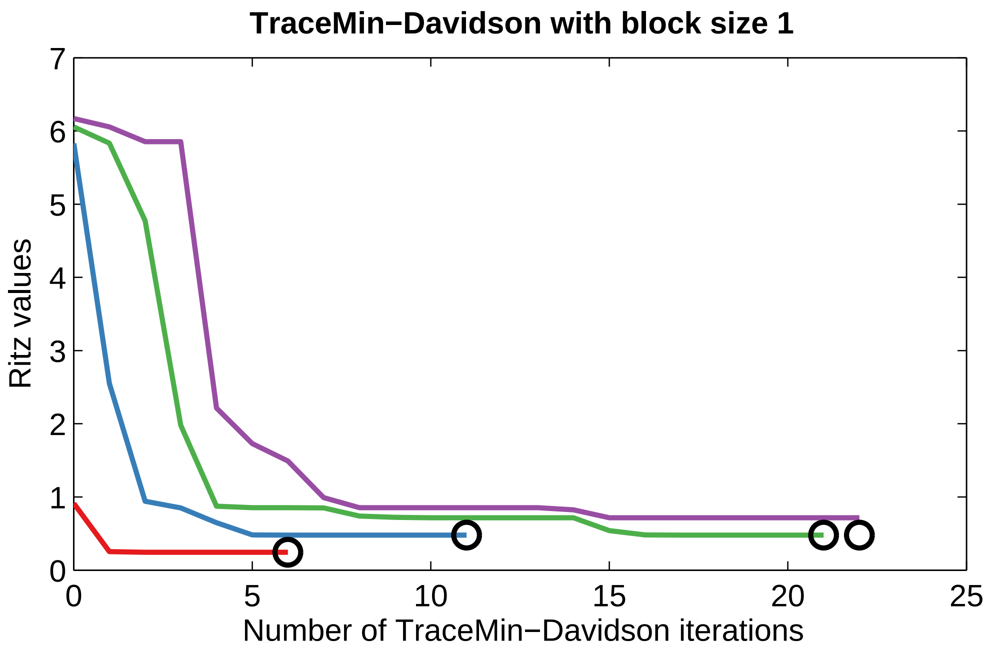
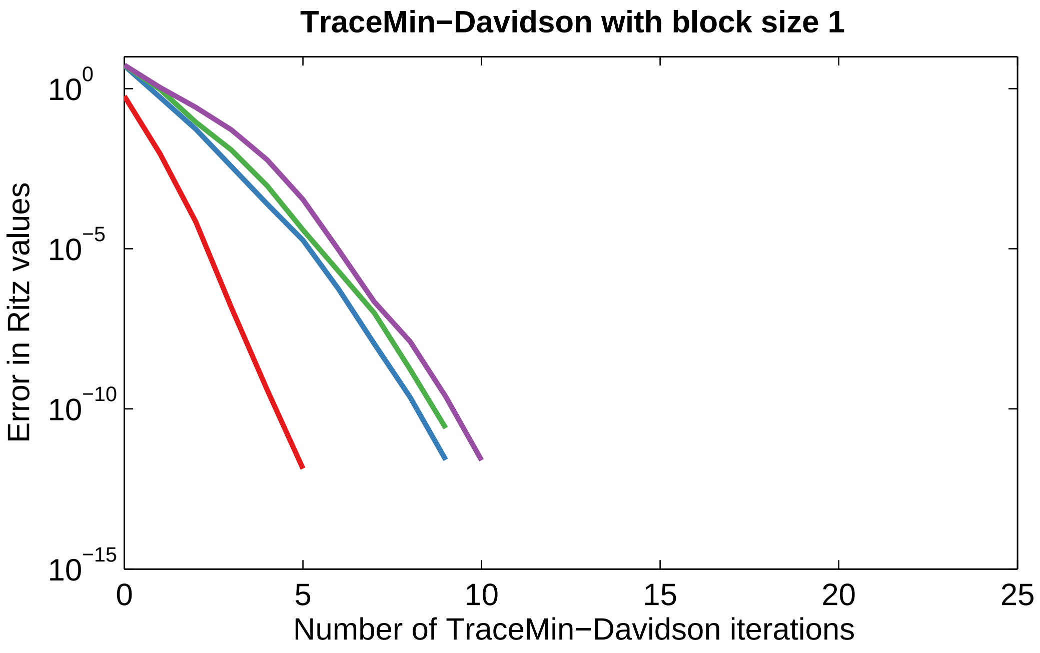


Figure 5.1.: The effect of block size on TraceMin-Davidson's convergence



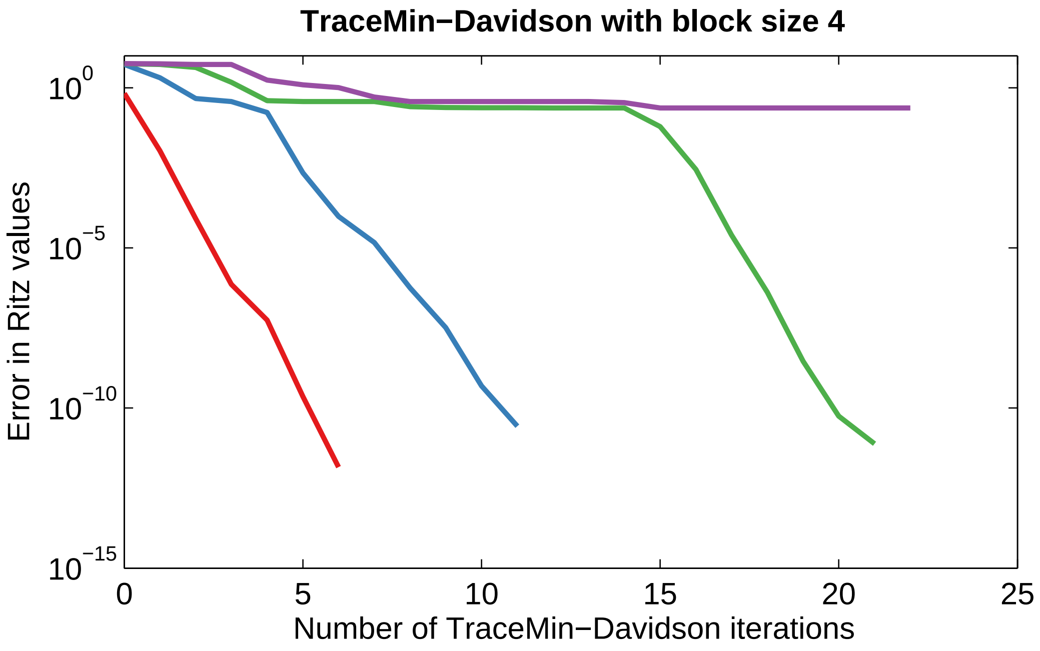


Figure 5.2.: The effect of block size on TraceMin-Davidson's convergence (continued)

and $\mathbb{L} = L^{-1}AL^{-T}\mathbb{K}$, where $B = LL^T$ is the Choleski factorization of B. We seek an approximate eigenvector y such that

$$y \in \mathbb{K} \tag{5.5}$$

and the Petrov-Galerkin condition

$$r \perp \mathbb{L} \tag{5.6}$$

As before, we have $y = Vu$ where u is the eigenvector corresponding to the desired eigenvalue of a slightly different eigenvalue problem

$$\hat{V}^T AB^{-1}A\hat{V}u = \theta\hat{V}^T A\hat{V}u \tag{5.7}$$

where $\hat{V} = L^{-T}V$; if this is a standard eigenvalue problem ($B = I$), we have

$$V^T A^2 Vu = \theta V^T AVu \tag{5.8}$$

instead. Assuming $W = L^{-1}AV$ is an orthonormal basis of $\mathbb{L}$, our problem can be rewritten as

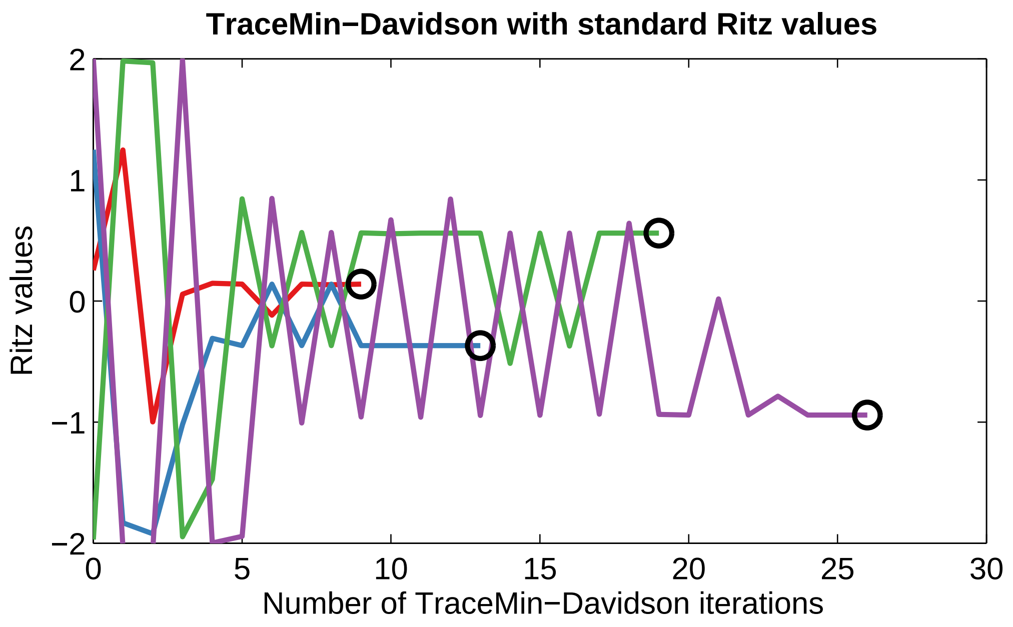
$$\hat{W}^T A^{-1}\hat{W}u = \frac{1}{\theta}u \tag{5.9}$$

(where $\hat{W} = LW$) making this method mathematically equivalent to using an orthogonal projection process for computing the eigenpairs of A^{-1}. The harmonic Ritz vectors maximize Rayleigh quotients for A^{-1}, so they can be interpreted as the best information one has for the smallest magnitude eigenvalues [19]. We now present a small problem demonstrating how the use of harmonic Ritz values can benefit TraceMin-Davidson.

We wish to compute the four smallest eigenpairs of an Anderson matrix of order $n = 64$ with an absolute residual $\|r\|_2 < 10^{-5}$; the eigenvalues of interest are (0.1391, -0.3688, 0.5609, -0.9419). We will use a block size of $s = 1$, an initial subspace of five vectors, and no restarts. Figure 5.3 shows that over time, we start converging to the correct eigenvalues (plotted as black circles) whether or not we compute the harmonic Ritz values. However, we see far more oscillation in the standard Ritz values than the harmonic ones. For instance, we appear to have “lost” the Ritz value approximating -0.9419 at iteration 21. That Ritz vector is still in the subspace; the corresponding Ritz value just isn’t the smallest. That may seem like a minor detail at first, but remember that when using an eigensolver with expanding subspaces, the order of the Ritz pairs matters. We are going to use the *first* Ritz vector to compute the next addition to the subspace. If the vectors were sorted poorly, we will not get a good addition to the subspace. When the Ritz values are sorted in such a way, we may face trouble computing Ritz shifts. We may even accidentally discard the Ritz vectors of interest upon restarting, which would be disastrous. By computing the harmonic Ritz values, we have changed how the Ritz pairs are sorted and see far less of this concerning oscillatory behavior.

5.4 Comparison of TraceMin and TraceMin-Davidson

Figure 5.6 presents a comparison of TraceMin and TraceMin-Davidson on the 3D discretization of the Laplace operator on the unit cube of order $n = 1000$. We would like to find the four smallest eigenpairs with an absolute residual of $\|r\|_2 < 10^{-5}$. TraceMin was run with a subspace dimension of $s = 8$, and TraceMin-Davidson added 8 vectors to the basis at each iteration. Both TraceMin and TraceMin-Davidson will solve a saddle point problem with 8 right hand sides at each iteration; the only difference in the amount of work required per iteration is in the Rayleigh-Ritz procedure, but this is very cheap in comparison to solving the saddle point problem. As figure 5.6 shows, TraceMin converged in 20 iterations, but TraceMin-Davidson only required 8. This is because TraceMin-Davidson was extracting its Ritz pairs from a larger subspace. In general, TraceMin-Davidson will converge in fewer iterations than TraceMin, but it does require far more storage than TraceMin.



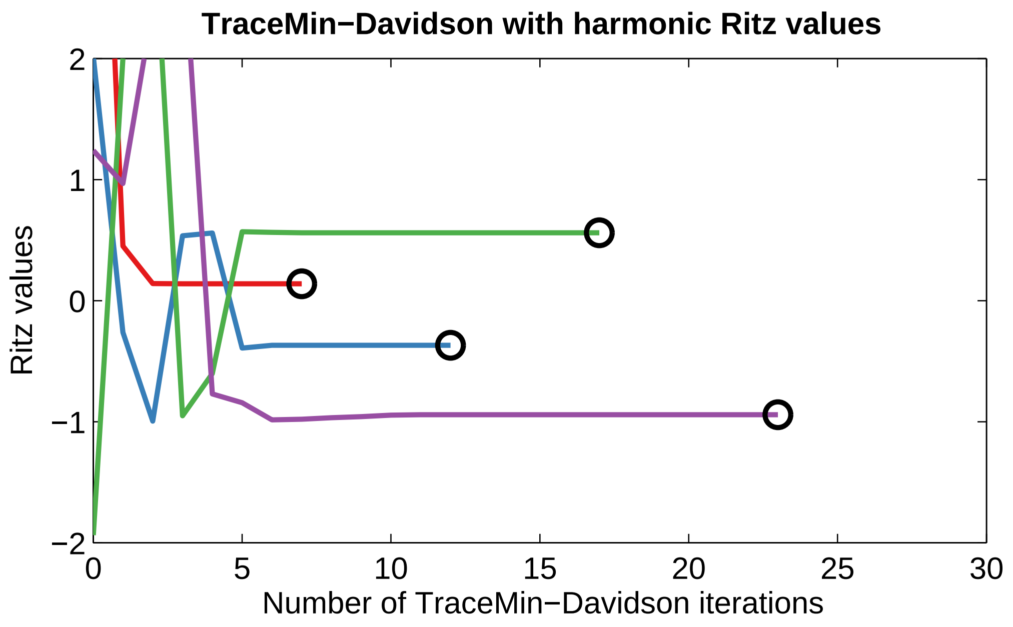
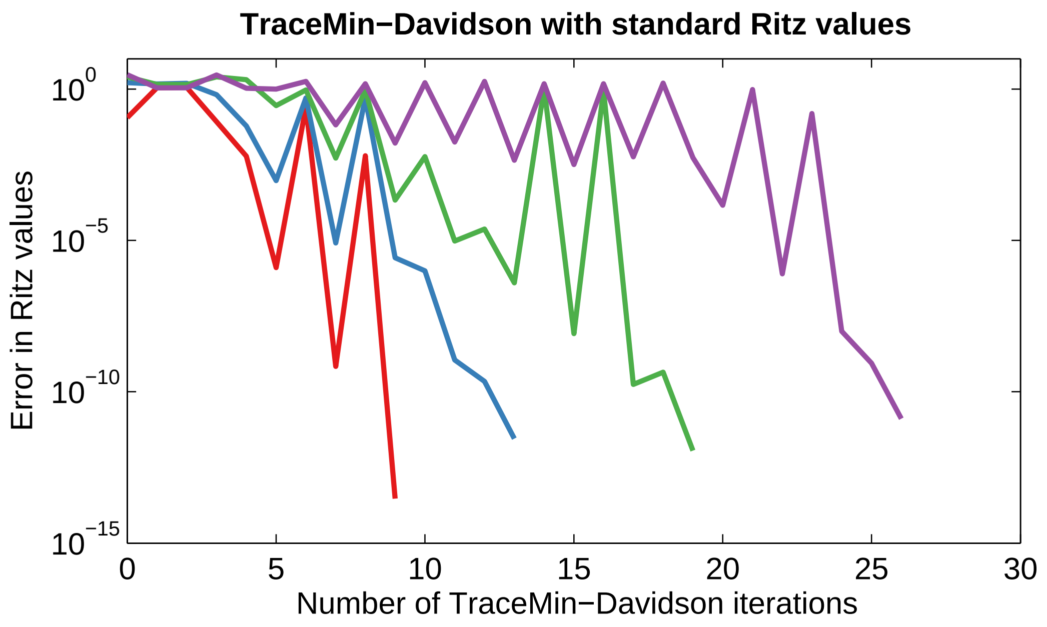


Figure 5.3.: A comparison of TraceMin-Davidson with standard and harmonic Ritz values



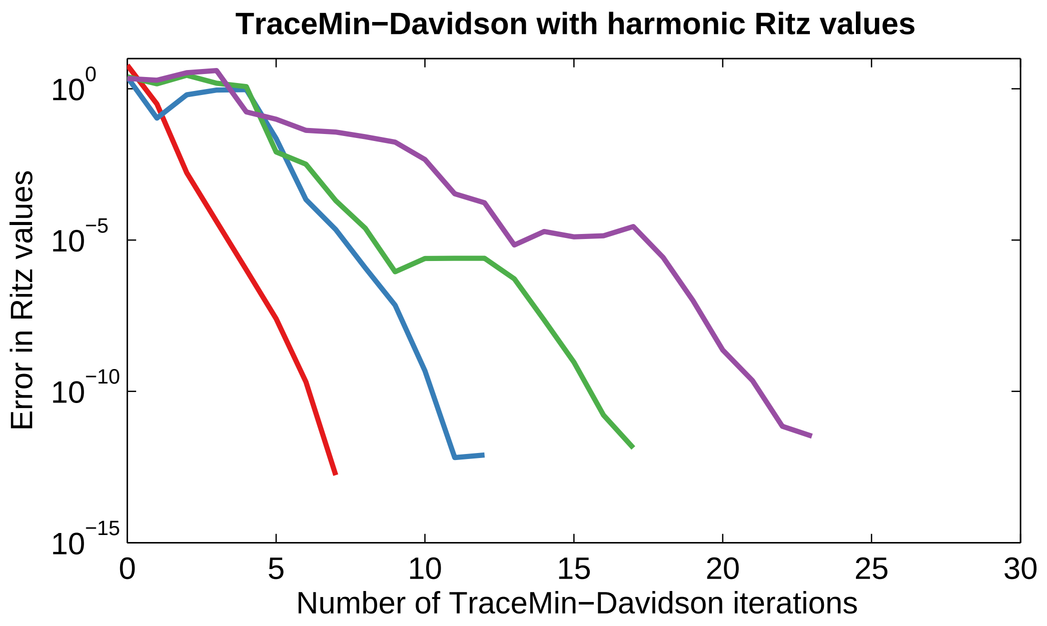
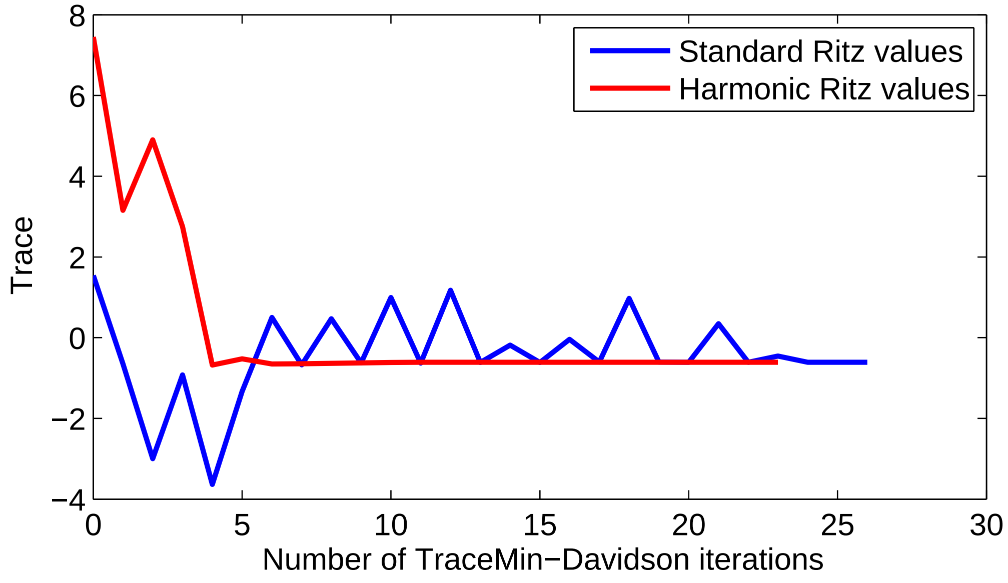


Figure 5.4.: A comparison of TraceMin-Davidson with standard and harmonic Ritz values (continued)



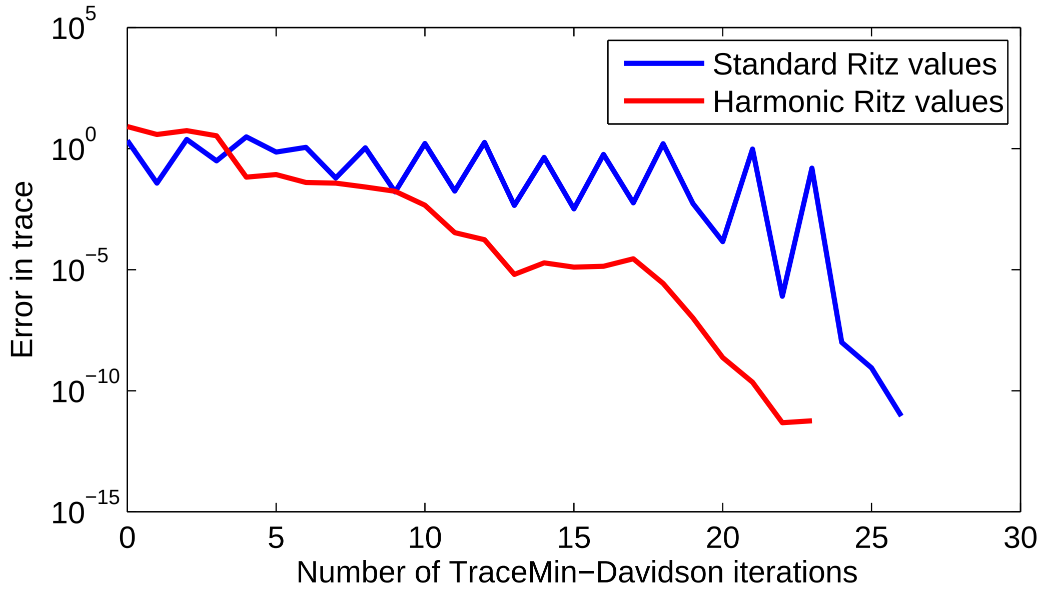


Figure 5.5.: A comparison of TraceMin-Davidson with standard and harmonic Ritz values (continued)

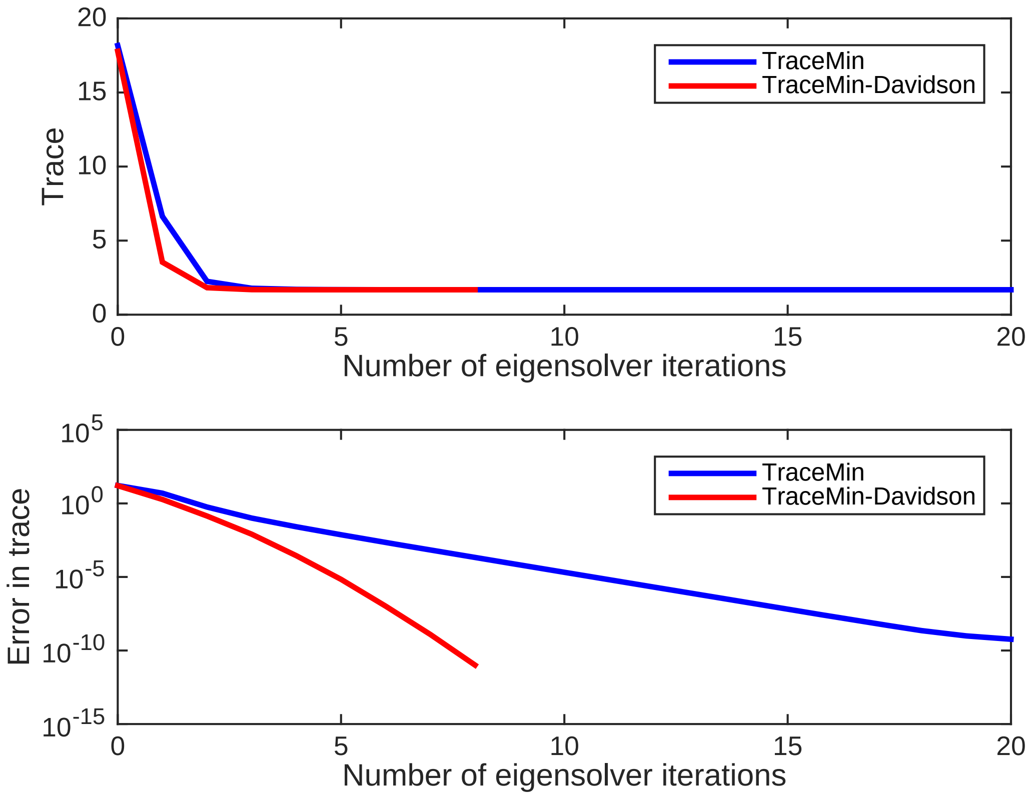


Figure 5.6.: A comparison of TraceMin and TraceMin-Davidson

6 IMPLEMENTATIONS

So far, we have discussed how to find a few of the smallest eigenpairs of the generalized eigenvalue problem $Ax = \lambda Bx$, but what if we have a different goal? For instance, we may want to compute a few of the largest eigenpairs, or we may wish to compute all of the eigenpairs within some interval. This chapte addresses how we modify TraceMin to solve various types of problems, divided into the following categories.

- Finding a very small number of interior or extreme eigenpairs (using TraceMin-Standard [1])
- Finding out whether there are any eigenvalues in a given interval, and finding a small subset of eigenpairs if they exist (using TraceMin-Sampling)
- Finding all eigenpairs in an interval, preferably one containing many eigenpairs (using TraceMin-Multisectioning)

6.1 Computing a few eigenpairs: TraceMin-Standard

In this section, we discuss how to use spectral transformations to compute a different target than the eigenpairs of smallest magnitude. We also discuss a special case known as the computation of the Fiedler vector.

6.1.1 Computing the eigenvalues of largest magnitude

If we wish to find the eigenvalues of largest magnitude (with their associated eigenvectors) of the problem

$$Ax = \lambda Bx \tag{6.1}$$

[1]Although we refer to this implementation as TraceMin-Standard, nothing would change if we used TraceMin-Davidson instead. This also applies to TraceMin-Sampling and TraceMin-Multisectioning.

where A and B are symmetric positive definite, this is equivalent to computing the smallest eigenvalues of

$$Bx = \sigma Ax \tag{6.2}$$

where $\sigma = \frac{1}{\lambda}$. We can run TraceMin on the problem of Equation 6.2 to obtain the solution of our target problem.

Note that if we wish to compute the largest eigenvalues of a standard eigenvalue problem ($B = I$), TraceMin's saddle point problem is greatly simplified. The solution to the saddle point problem is

$$V = AY \left(Y^T A^2 Y\right)^{-1} \tag{6.3}$$

All that is required to solve this problem is an inner product and the solution of a small dense linear system with many right hand sides[2]. We do not need to solve a single linear system of the form $Ax = b$.

If we wish to compute the largest eigenvalues of a standard eigenvalue problem, Ritz shifts should be disabled, since they would require solving linear systems of the form $(I - \omega A)\, x = b$, where ω is our desired shift. Even if the eigenvalue distribution is poor, performing many iterations without the shift will probably be cheaper than solving the linear systems required by the use of a shift.

6.1.2 Computing the eigenvalues closest to a given value

If we would like to compute the eigenpairs nearest a given value α, we need only solve the eigenproblem

$$(A - \alpha B)\, x = (\lambda - \alpha)\, Bx \tag{6.4}$$

If $(\lambda - \alpha, x)$ is an eigenpair of (6.4), then (λ, x) is an eigenpair of the original problem $Ax = \lambda Bx$.

[2]Technically, we do not need to solve those small dense linear systems because AY and $AY\left(Y^T A^2 Y\right)$ lie in the same subspace.

6.1.3 Computing the absolute smallest eigenvalues

If we wish to compute the absolute smallest eigenvalues of a matrix, we may simply shift to the left edge of the spectrum and run TraceMin as we normally would. However, this requires us to be able to bound the eigenvalues. We can use the Gerschgorin circle theorem to determine some β such that $A - \beta B$ is symmetric positive semi-definite and compute the smallest magnitude eigenpairs of 6.4. The β produced by the Gerschgorin circle theorem tends to be far from the smallest eigenvalue, which means TraceMin will have a lackluster convergence rate, but we have already studied how dynamic Ritz shifts can help solve this problem.

6.1.4 Computing the Fiedler vector

In this case, we are interested in the eigenvector corresponding to the smallest nonzero eigenvalue of a standard eigenvalue problem $Ax = \lambda x$, where A is symmetric positive semi-definite. This eigenvector, known as the Fiedler vector, tells us how to reorder a matrix to either reduce the bandwidth or bring large elements toward the diagonal. If the graph consists of only one strongly connected component[3], the graph Laplacian A's null space is exactly one vector, the vector of all 1s[4].

The fact that A is singular can cause problems for some eigensolvers. However, we have already demonstrated that TraceMin does not rely on accurate linear solves. Furthermore, if we project this null vector out of the set of basis vectors V_k at each TraceMin iteration, all of the linear systems we solve will be consistent.

[3]If the graph consists of multiple strongly connected components, it can be split up into many smaller eigenproblems, one per strongly connected component.

[4]We assume for simplicity that A was not normalized.

6.1.5 Computing interior eigenpairs via spectrum folding

If we wish to compute the interior eigenpairs of a standard eigenvalue problem $Ax = \lambda x$, we may instead solve the equivalent eigenvalue problem

$$A^2 x = \lambda^2 x \tag{6.5}$$

This is known as *spectrum folding*[5]. Instead of seeking the smallest magnitude eigenpairs of the symmetric indefinite matrix A, i.e. the eigenvalues closest to zero (which can be positive or negative), we seek the smallest magnitude eigenpairs of the symmetric positive definite operator A^2. Note that it is a bad idea to form the matrix A^2 explicitly, so we apply that operator implicitly. Working with this operator has several side effects. The most obvious side effect is that we have squared the condition number of the matrix, so it is now more difficult to solve the linear systems arising at each TraceMin iteration. Again, TraceMin does not rely on accurate linear solves, so this should not impede TraceMin's convergence as much as other eigensolvers. The other effect of this transformation is that instead of each eigenpair converging at the rate

$$\frac{\lambda_i}{\lambda_{s+1}} \tag{6.6}$$

they now converge at the faster rate[6]

$$\left(\frac{\lambda_i}{\lambda_{s+1}}\right)^2 \tag{6.7}$$

Also note that instead of solving a linear system of the form $A^2 x = b$ at each TraceMin iteration, we can instead solve two linear systems each of the form $Ax = b$.

[5]TraceMin and TraceMin-Davidson are both capable of computing interior eigenpairs without using spectrum folding, but the trace-minimization property no longer holds. If we use spectrum folding, TraceMin's global convergence proof still holds.

[6]This convergence rate helps us to estimate how many iterations TraceMin would require if the linear systems were solved directly. When we use an iterative solver, we will likely need more TraceMin iterations to converge.

To demonstrate the effect of spectrum folding, let us again solve the Anderson problem of section 5.3. We will compute the four smallest eigenpairs of that same Anderson matrix of order $n = 64$ with an absolute residual $\|r\|_2 < 10^{-5}$ using a block size of $s = 1$, an initial subspace of five vectors, and no restarts. As a reminder, the eigenvalues of interest are (0.1391, -0.3688, 0.5609, -.9419). This time, we will try spectrum folding to compute the eigenpairs of smallest magnitude. Figure 6.1 shows that whether we used the standard Ritz extraction, harmonic Ritz extraction, or spectrum folding, we converged to the correct eigenvalues. Using a harmonic Ritz extraction, TraceMin-Davidson took 23 iterations to converge; with spectrum folding, we only required 17 TraceMin-Davidson iterations (since the spectrum of eigenvalues was improved). Because the matrix was so small, we formed A^2 explicitly for spectrum folding and used a direct solver to solve the saddle point problem arising at each TraceMin-Davidson iteration. As a result, spectrum folding would have been the fastest method in this case. For larger problems where that is infeasible, spectrum folding will require roughly twice as much work per TraceMin-Davidson iteration (as compared to not using spectrum folding on that same problem). If spectrum folding reduces the number of required TraceMin-Davidson iterations by a factor of two or greater, it will be effective; otherwise, it may result in a longer running time.

Although it may appear in figures 6.1, 6.2, and 6.3 that the trace is not decreasing monotonically in the case of spectrum folding, that is only because the trace presented is the trace of X^TAX. In spectrum folding, the trace of X^TA^2X is being decreased monotonically (as figures 6.4 and 6.5) demonstrate. When the Rayleigh quotients $x_i^TA^2x_i$ become very close to λ_i^2 (the square of the true eigenvalues), the Rayleigh quotients $x_i^TAx_i$ approach λ_i as well.

6.1.6 Our parallel TraceMin-Standard implementation

We chose to write our TraceMin and TraceMin-Davidson implementations using the Trilinos framework; our code is publicly available and can be downloaded from

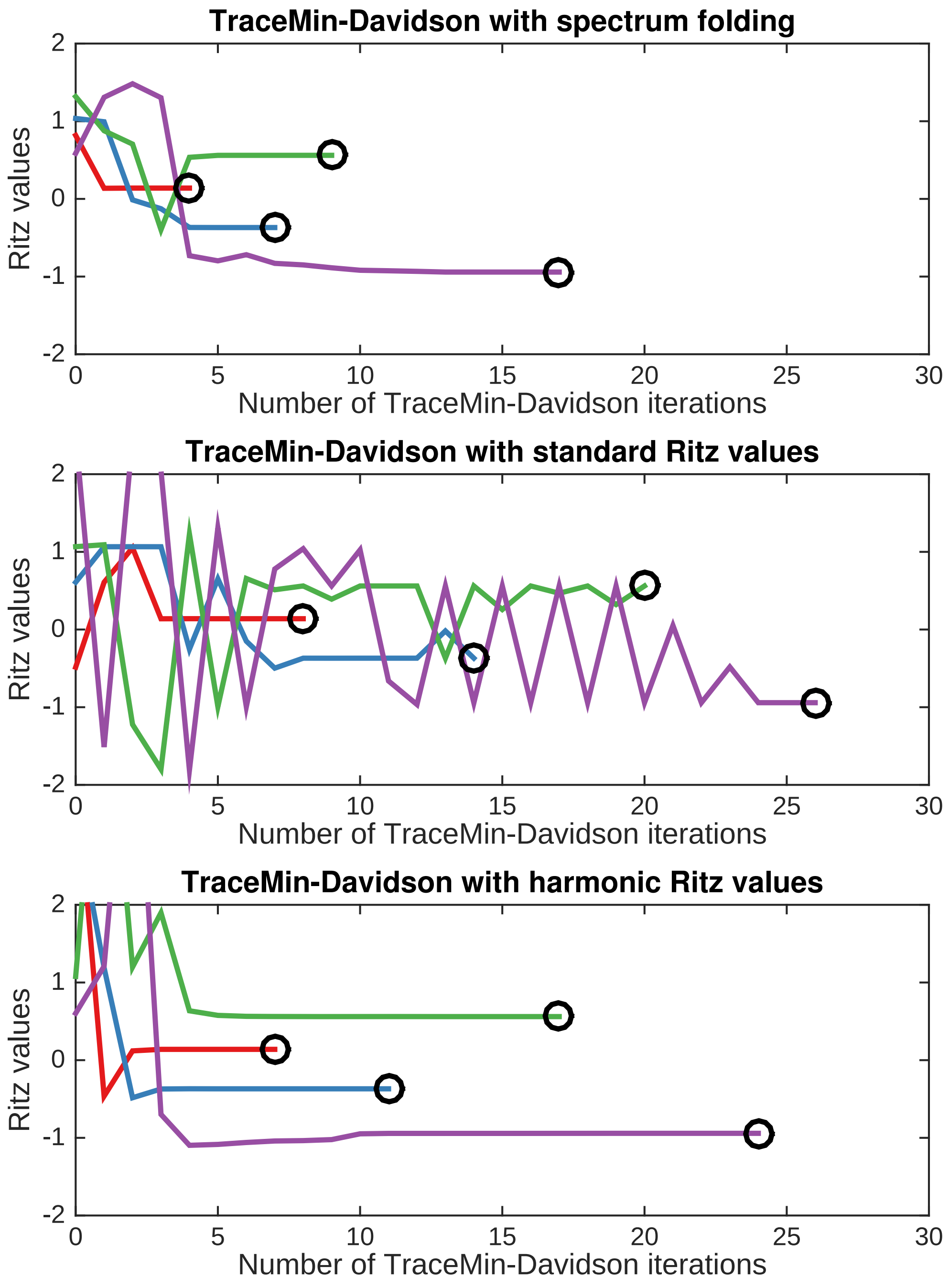


Figure 6.1.: A demonstration of the effect of spectrum folding

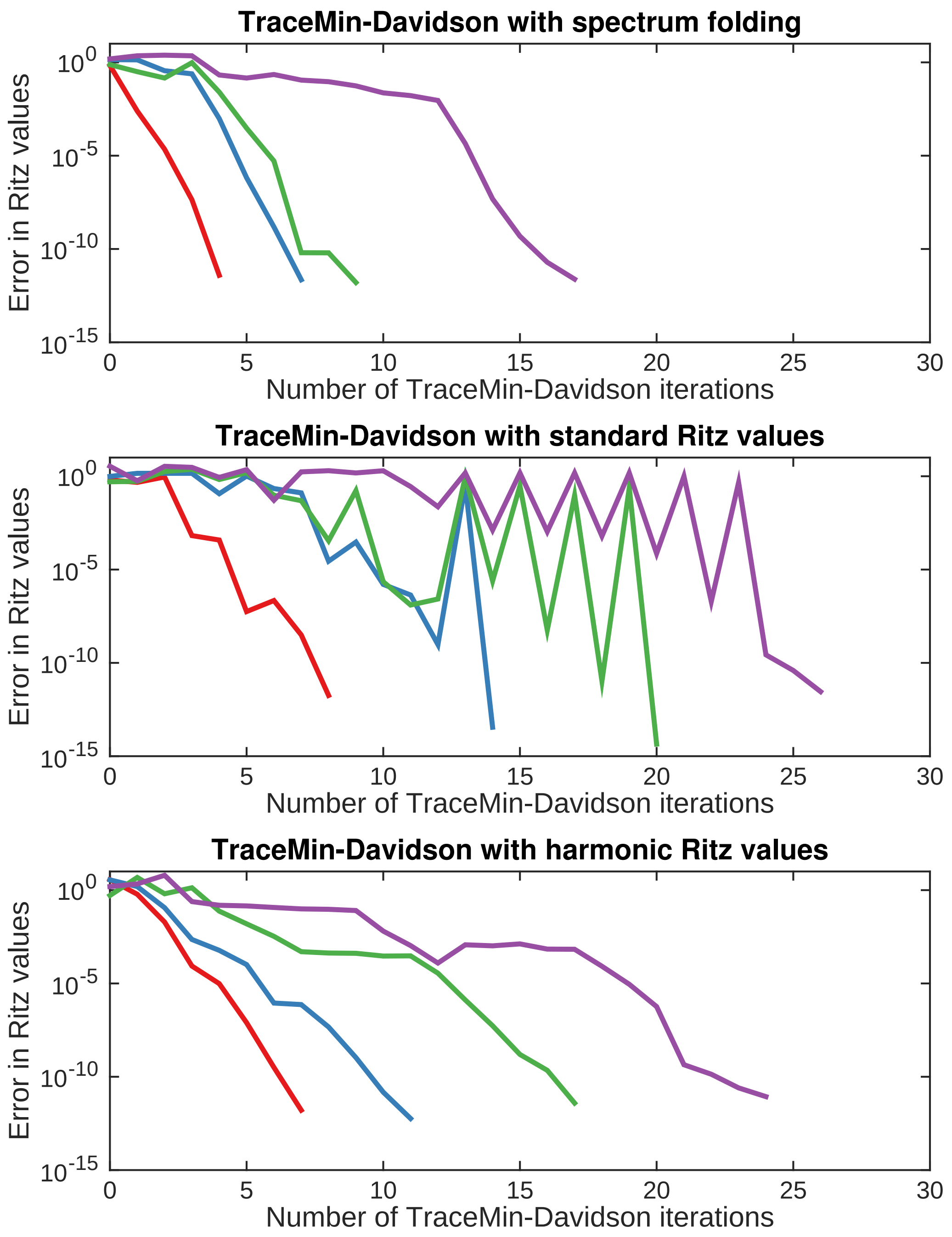
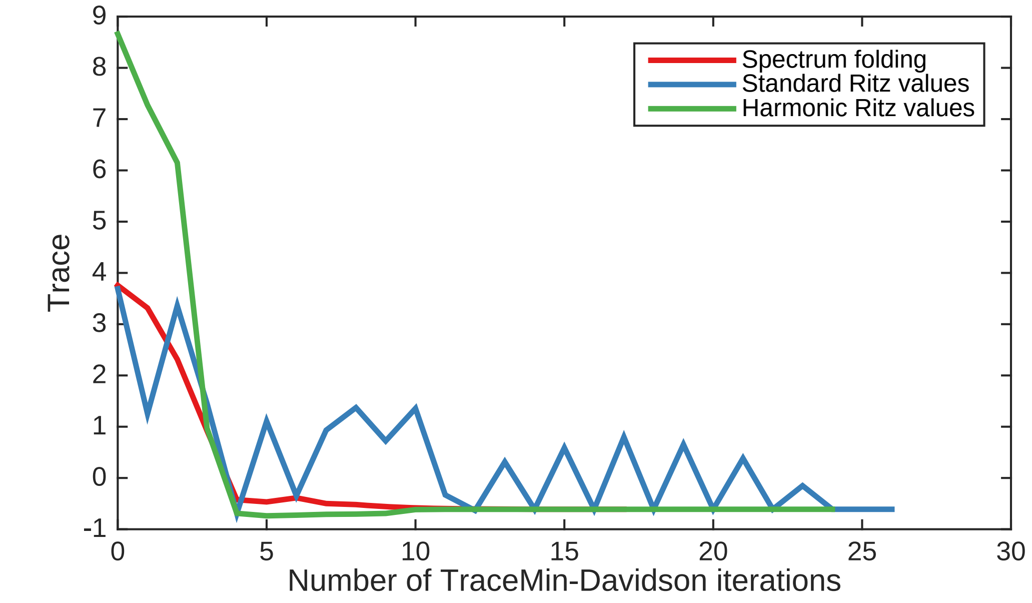


Figure 6.2.: A demonstration of the effect of spectrum folding (continued)



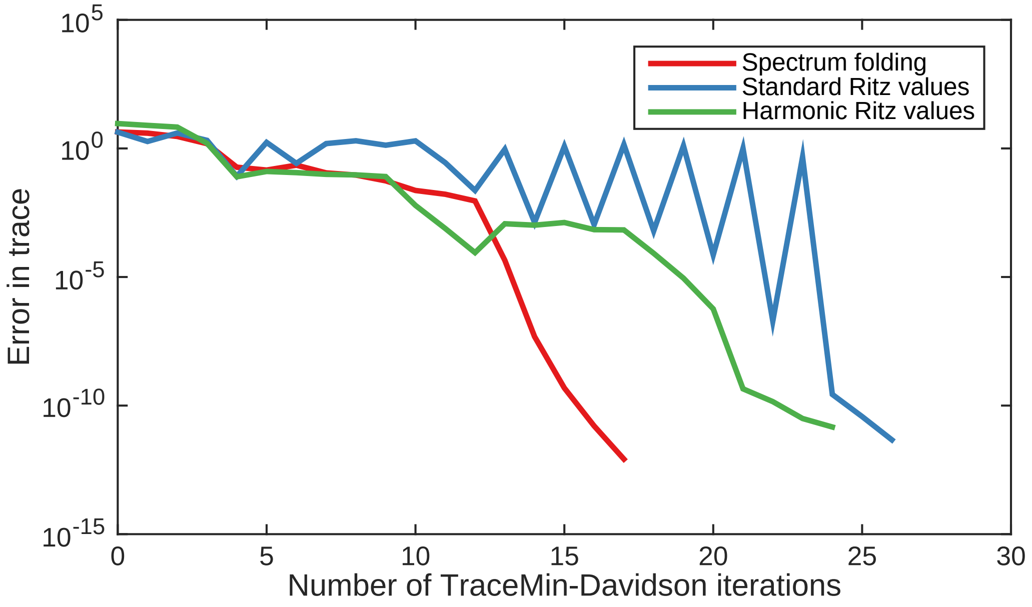
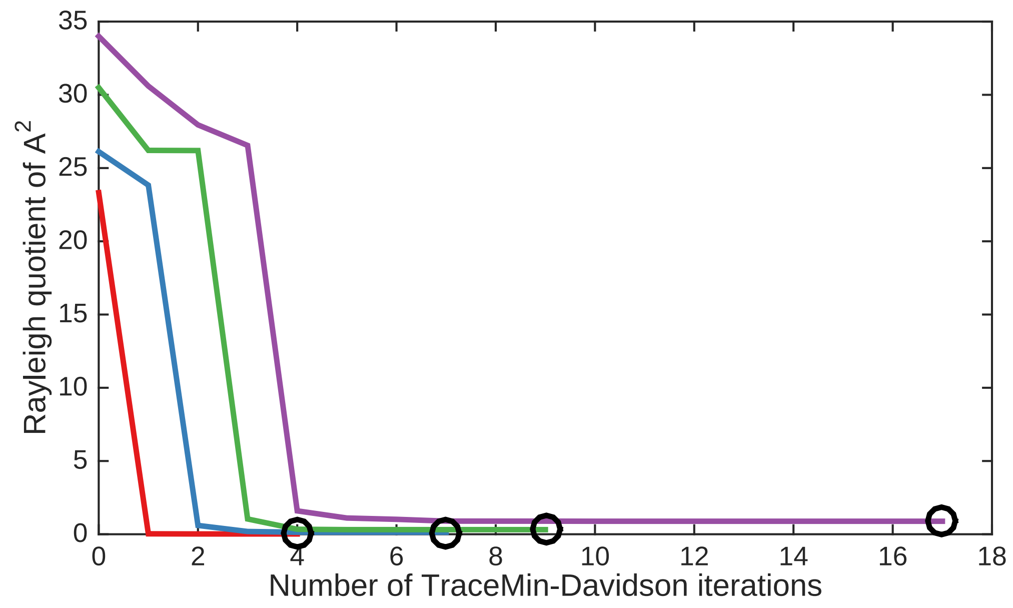


Figure 6.3.: A demonstration of the effect of spectrum folding (continued)



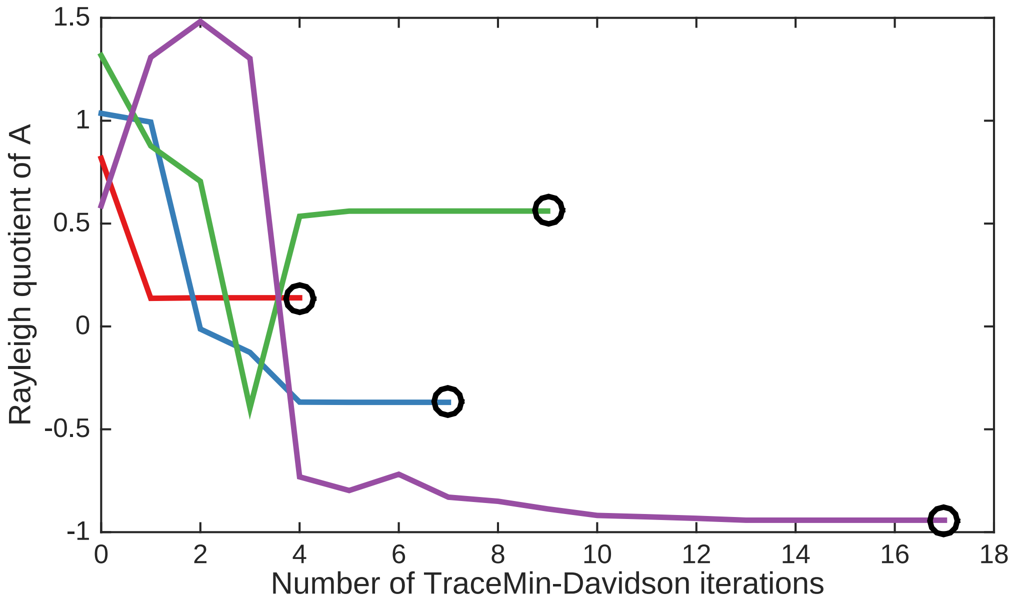
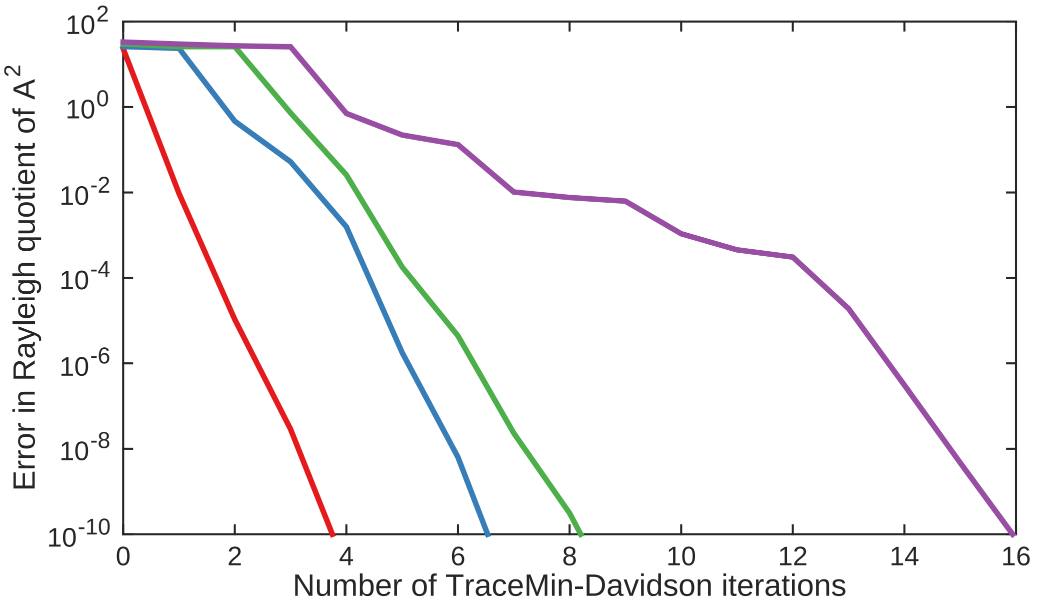


Figure 6.4.: A demonstration of the effect of spectrum folding (continued)



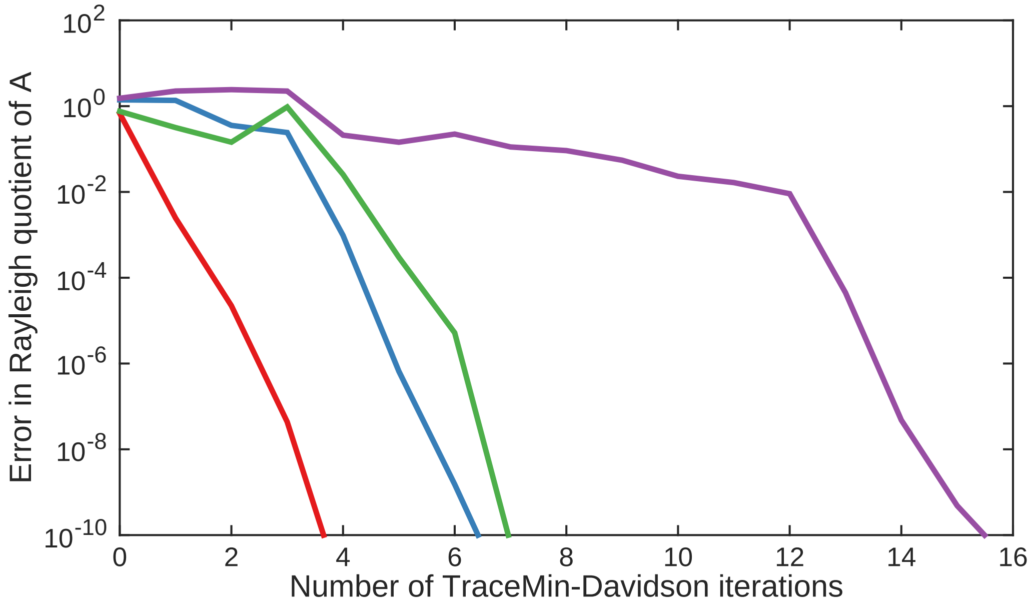


Figure 6.5.: A demonstration of the effect of spectrum folding (continued)

the Trilinos website [20]. Trilinos contains a variety of linear system and eigenvalue problem solvers (among other things) similar to PETSc/SLEPc [21–26], but unlike PETSc, Trilinos supports block linear solves. Trilinos has impressive parallel scalability, supports very large problems, and is effective on many different architectures including GPUs. To make the spectral transformations of this section easier for the user, we provided several example drivers demonstrating their use, all of which are available in the Trilinos Doxygen documentation [27].

The sparse matrices A and B are stored in compressed sparse row format, using a block row distribution [7]. Tall dense matrices such as V and Y (referred to in Trilinos as multivectors) are stored using a block row distribution as well. Small dense matrices such as $H = V^T AV$ are replicated rather than distributed; each MPI process owns a copy. Using this data distribution, the following distributed kernels are required

- sparse matrix times multivector multiplication (referred to as a matvec)
- inner product of two multivectors
- B-orthonormalization of a multivector
- solution of a sparse symmetric linear system with multiple right hand sides

Within a node, we can choose to use OpenMP for shared memory parallelism, CUDA for GPUs, or we can simply spawn more MPI processes[8].

Our Trilinos implementation of TraceMin uses the matvec, inner product, and B-orthnormalization routines defined in Trilinos, although it also allows users to provide their own implementations of any of these operations [9]. The Tpetra matvec

[7]This is how we chose to store the matrices for the test cases we will present later, but it is not a requirement.

[8]Trilinos allows its users to specify a node type and switches its strategy for handling shared memory computations based on that node type.

[9]Our TraceMin implementation only needs to know how a matrix multiplication works, as well as how certain vector operations are performed; it does not need the matrices or vectors to be stored in any specific way. This allows users to take advantage of any special structure their matrices might have, or write code that performs well on unique architectures. It also means that TraceMin can be used to solve problems where the matrix is never made explicitly available.

has been optimized to take advantage of the sparsity pattern of the matrix, and it performs the minimum amount of communication required. Trilinos also contains three different orthonormalization routines which can be used in parallel: modified Gram-Schmidt [28], tall skinny QR [29, 30], or by using algorithm 6 which uses the eigendecomposition of a small dense matrix to B-orthonormalize a set of vectors.

The Belos package of Trilinos contains many Krylov solvers capable of solving linear systems with multiple right hand sides. Some of these are block methods which build one shared Krylov subspace for all right hand sides, and others are pseudo-block, meaning they are mathematically equivalent to solving each linear system independently, but the communication and memory accesses are more effiecient. Block methods can solve sets of linear systems of the form $Ax_i = b_i$, where all right hand sides b_i are available simultaneously. In the case of TraceMin, we end up having to solve linear systems of the form $(A - \omega_i B)\, x_i = b_i$ if we choose to use multiple dynamic Ritz shifts. Belos' pseudoblock Krylov solvers are currently incapable of solving linear systems with indexed operators such as we have here, so we wrote our own pseudo-block MINRES which accepts indexed operators. This MINRES uses the efficient parallel kernels we mentioned previously.

Algorithm 6 Orthnormalization via eigendecomposition

```
Require: $B \in \mathbb{R}^{n \times n}$ symmetric positive definite
    $V_{old} \in \mathbb{R}^{n \times s}$ with rank $s$ is the set of vectors to be $B$-orthonormalized
1: Form $H_2 = V_{old}^T B V_{old}$
2: Compute the eigendecomposition of $H_2$, $H_2 X_2 = X_2 \Theta_2$
3: Form $V_{new} = V_{old} X_2 \Theta_2^{\frac{-1}{2}}$, which is $B$-orthonormal
```

6.2 TraceMin-Sampling

We are now interested in finding out whether any eigenvalues exist within a certain interval. If so, we must obtain a few eigenpairs near a set of shifts within that interval. These shifts can be handled completely independently.

We spawn a number of MPI processes equal to the number of desired shifts. Assuming sufficient space is available, matrices A and B are replicated on each node. Then we run a separate instance of TraceMin on each node, using a spectral transformation to find the eigenpairs nearest a given shift. This process requires *no* communication across nodes. The only thing hindering parallel scalability is the potential for load imbalance.

Note that while this choice of MPI processes is optimal from a scalability standpoint, it is by no means required. If it is infeasible to replicate A and B on each node, we may also divide our MPI processes into small groups, perhaps 4 processes per group. Then, each group of processes would store the matrices in a distributed fashion. Instead of running one instance of TraceMin per MPI process, we could then run one instance of TraceMin per group. There would then be a small amount of MPI communication, but it would be limited to the processes within the individual groups. There would be no global communication required.

If the number of desired shifts is greater than the number of groups of MPI processes, each group would be assigned a small subset of shifts and run TraceMin once for each shift. A potential load balancing strategy for such a case is explored in the next section.

6.3 TraceMin-Multisectioning

In this case, we want to find *all* the eigenpairs within a given interval (which we refer to as the global interval). Assuming this interval contains many eigenpairs, it would be impractical to run a single instance of TraceMin to compute all the eigenpairs together; we might not even have enough space to store all of the required eigenvectors. We need to break the interval up into smaller pieces, each of which can be solved independently.

We propose a method similar in nature to adaptive quadrature. In adaptive quadrature, you want to calculate the integral of some function on a given interval.

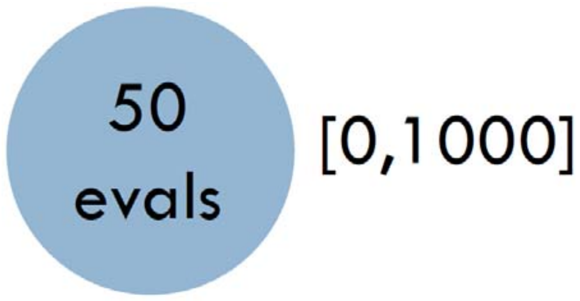
If the interval is "bad," namely the error estimate is too large, then you break it in two and repeat the process with each half. You continue recursing in this way until you have a set of satisfactory intervals; whether an interval is satisfactory is defined by a tolerance parameter.

In the case of TraceMin-Multisectioning, we start with some global interval of interest just like in adaptive quadrature. Then, we evaluate whether the interval is "bad," meaning it contains too many eigenvalues. If so, it is divided in half and the procedure is repeated. This process continues recursively until we have a set of satisfactory intervals; each interval must contain at most n_e eigenvalues, where n_e is a parameter defined by the user. Figure 6.6 illustrates the multisectioning procedure.

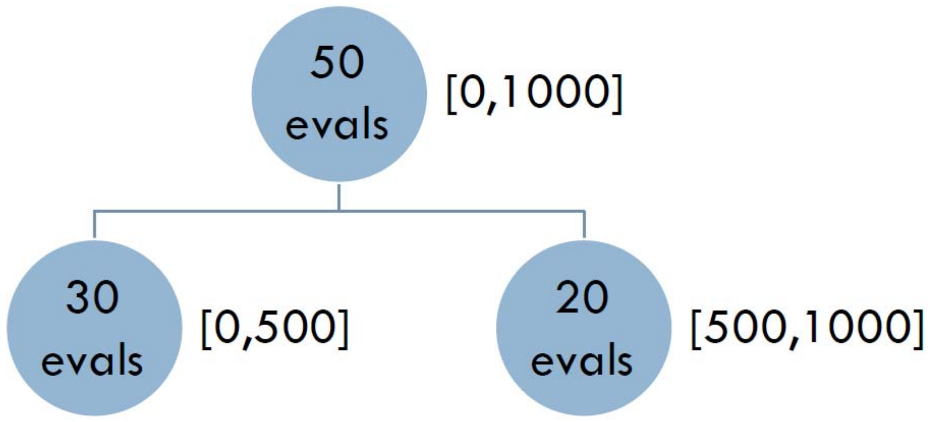
Let us assume an interval containing 20 eigenvalues is sufficiently small (i.e. $n_e = 20$). In the first image, we start with the interval containing all eigenvalues in the range $[0, 1000]$. We know there are 50 eigenvalues in that interval. Since that is too many, we divide the interval in half and obtain two smaller intervals: $[0, 500]$, which contains 30 eigenvalues, and $[500, 1000]$, which contains 20. The second subinterval is sufficiently small and does not need to be subdivided further. The first one, however, gets divided into the intervals $[0, 250]$ and $[250, 500]$, each of which contain fewer than 20 eigenvalues. In the end, rather than running TraceMin on the interval $[0, 1000]$, we run 3 independent instances of TraceMin on the intervals $[0, 250]$, $[250, 500]$, and $[500, 1000]$.

6.3.1 Obtaining the number of eigenvalues in an interval

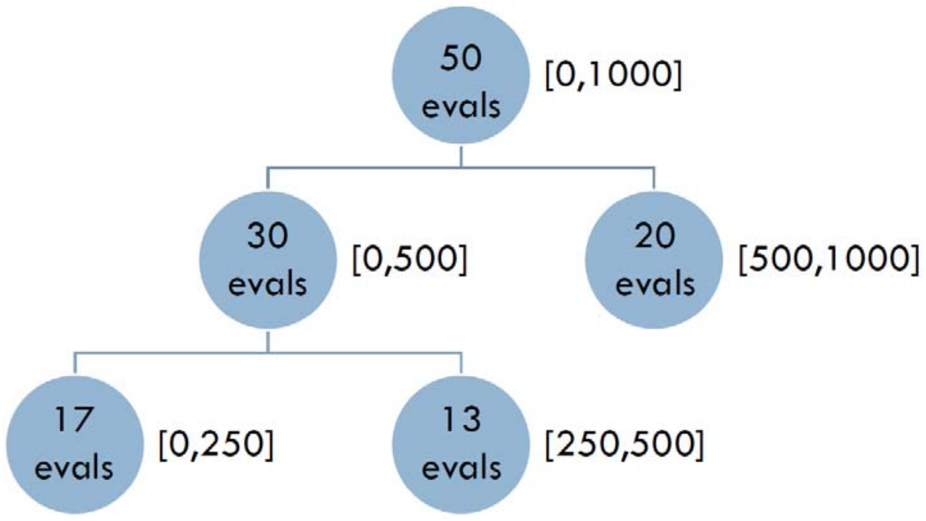
To obtain the number of eigenvalues in a particular interval, we use a sparse factorization method such as PARDISO [31,32], MUMPS [33,34], or WSMP to compute the inertia of a shifted matrix [35]. If $A - aB$ has p_1 positive eigenvalues, and $A - bB$ has p_2 positive eigenvalues, then $Ax = \lambda Bx$ has $p_1 - p_2$ eigenvalues in the interval (a, b).



(a) Initial interval



(b) After one subdivision



(c) Final result

Figure 6.6.: An example of interval subdivision for multisectioning

We now turn our attention to how these small intervals should be assigned to various MPI processes.

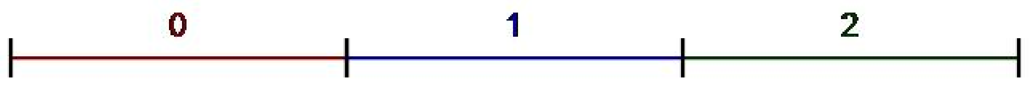
6.3.2 Assigning the work

This section describes our two implementations of TraceMin-Multisectioning and how they divide the work.

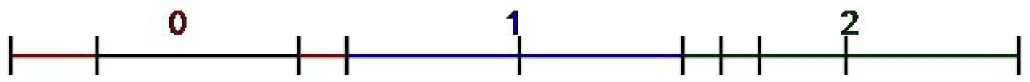
Static work allocation

One way to implement this is to use a static work allocation. Each MPI process is assigned a segment of the large global interval to subdivide (as in Figure 6.7a). Figure 6.7b shows the result of our subdivision: each MPI process now owns a set of small intervals. Some intervals may turn out to be empty and get discarded; the black line under process 0 denotes an empty interval that got discarded. Each of these MPI processes now has a different amount of work, so we perform one communication where the work gets redistributed so that each MPI process is given a roughly equal number of subintervals on which to run TraceMin. The redistributed work is shown in Figure 6.7c.

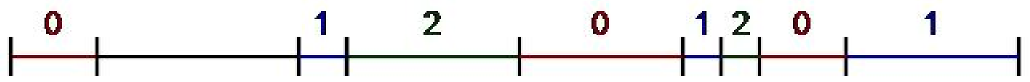
This implementation has the advantage of requiring absolutely no communication after the work has been divided amongst the processes. However, there exists the potential for a high load imbalance for two reasons. First of all, the number of subintervals may not be evenly divisible by the number of processes. For instance, if we obtain five subintervals from the recursive division of the large global interval with four MPI processes, one process will be assigned twice as many intervals as the others. More importantly though, the number of assigned intervals is not a good estimate of the amount of work, since different intervals may require vastly different amounts of work. One would expect intervals containing more eigenvalues to be more computationally intensive, but the factor that most greatly influences the running time is the distribution of eigenvalues in each interval, which is unknown until we've



(a) Each MPI process has one part of the interval to subdivide



(b) Each MPI process has subdivided their initial interval and now owns several smaller intervals



(c) The final work distribution

Figure 6.7.: An example of static work allocation for TraceMin-Multisectioning with 3 MPI processes

run several iterations of TraceMin. Even if every MPI process were assigned the same number of subintervals, and each subinterval contained the same number of eigenvalues, there would still be potential for a massive load imbalance simply because some intervals have a considerably more favorable eigenvalue distribution than others.

Dynamic work allocation

To remedy the load imbalance issues of the previous implementation, we can dynamically assign the work as needed. This process is best described via an analogy.

At McDonalds (or any other large company), there is a hierarchical structure to the employees. There is one CEO who is responsible for assigning work to the other employees. That is his entire responsibility; he doesn't go down to the kitchen and flip burgers. McDonalds also employs thousands of workers who are only responsible for doing the work, i.e. flipping burgers. These workers never communicate with the CEO directly because that would be overwhelming for the CEO. Instead, they are divided up into groups based on their location, and each group has a store manager.

The store manager is responsible for relaying important messages between the workers and the CEO. Unlike the CEO, the store manager is also required to flip burgers.

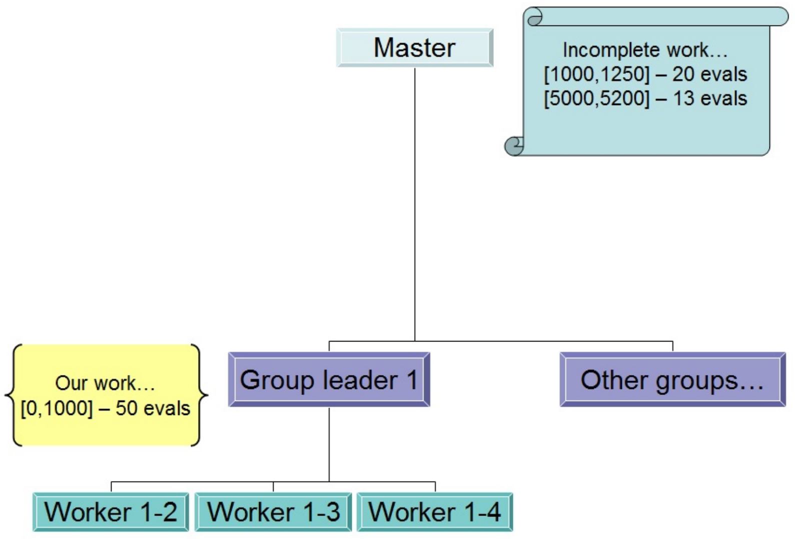
In this TraceMin implementation, we divide the MPI processes into three general categories similar to the categories of McDonalds employees:

- *master*: similar to the CEO, responsible for assigning work
- *worker*: similar to the burger-flippers, responsible for doing work on an individual interval
- *group leader*: similar to the store manager, responsible for work and communication

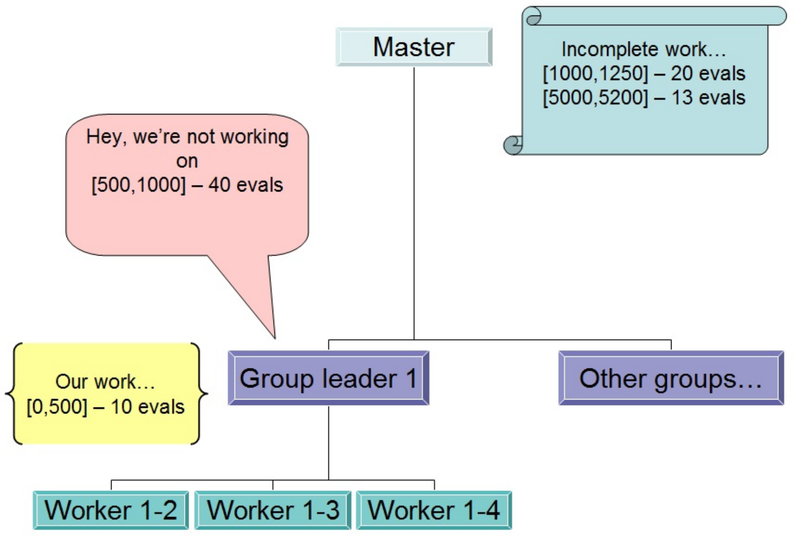
The MPI processes are broken up into groups consisting of a leader and many workers. Each group handles one interval at a time (which has been assigned by the master). There are two types of communication for two levels of parallelism. The master sends messages to the individual group leaders, informing them of which subinterval their group is expected to process. The group leader and workers collaborate to run TraceMin on their assigned subinterval. Figure 6.8 illustrates this process with an example.

Which method to choose

With very few MPI processes, it does not make sense to use a dynamic work allocation, since it prevents one MPI process from doing any useful work; it is preferable to use the static work allocation in that case. If there are enough MPI processes for the load imbalance to become apparent, the dynamic work allocation works better.

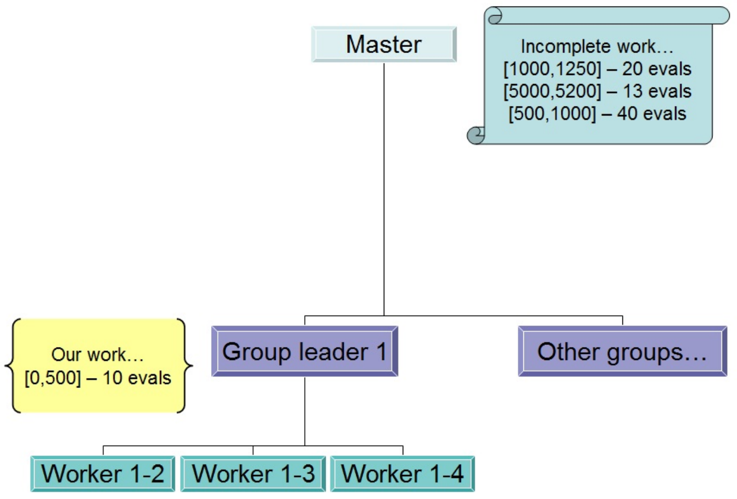


(a) The master maintains a list of work that still needs to be done. Group 1 has a large interval that must be subdivided.

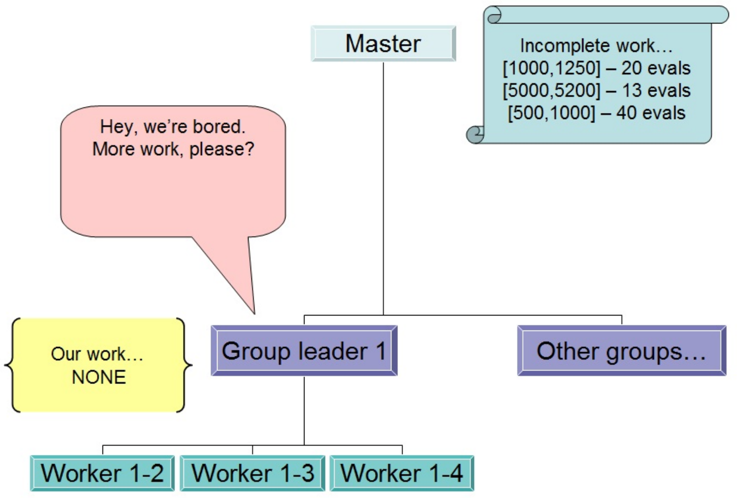


(b) Group 1 performs a factorization to divide its interval in two. It keeps one of the subintervals, and the group leader sends the other to the master.

Figure 6.8.: A demonstration of TraceMin's dynamic load balancing

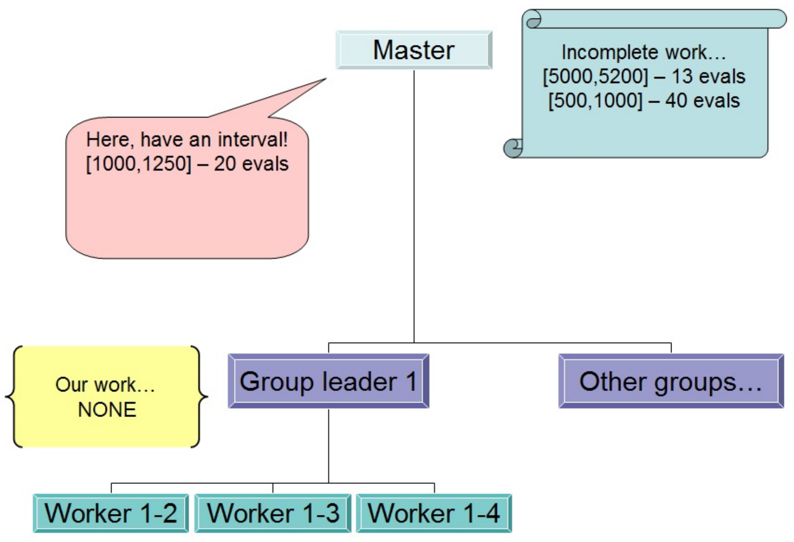


(a) The master has added the subinterval (500,1000) to the list of incomplete work. Meanwhile, group 1 runs TraceMin on the small interval (0,500).

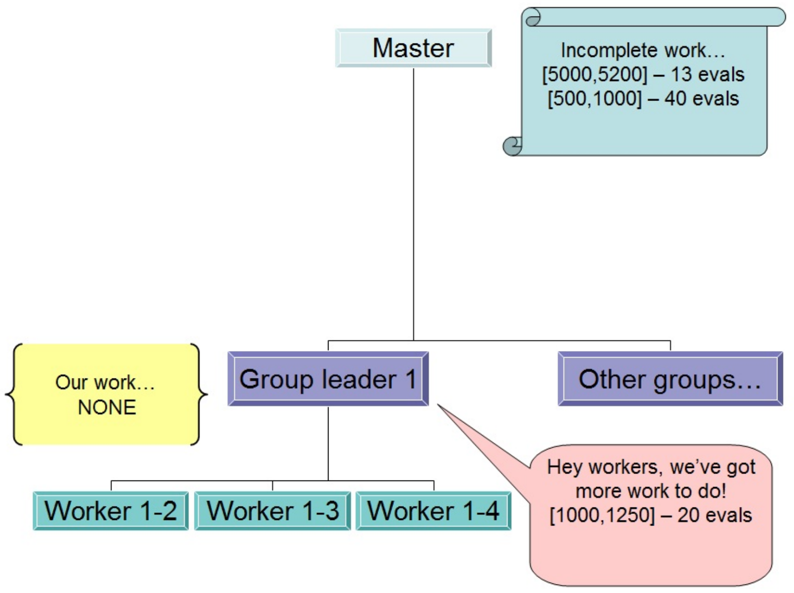


(b) Group 1 has found the 10 eigenvalues in the interval (0,500) and needs more work. The group leader requests more work from the master.

Figure 6.8.: A demonstration of TraceMin's dynamic load balancing (continued)

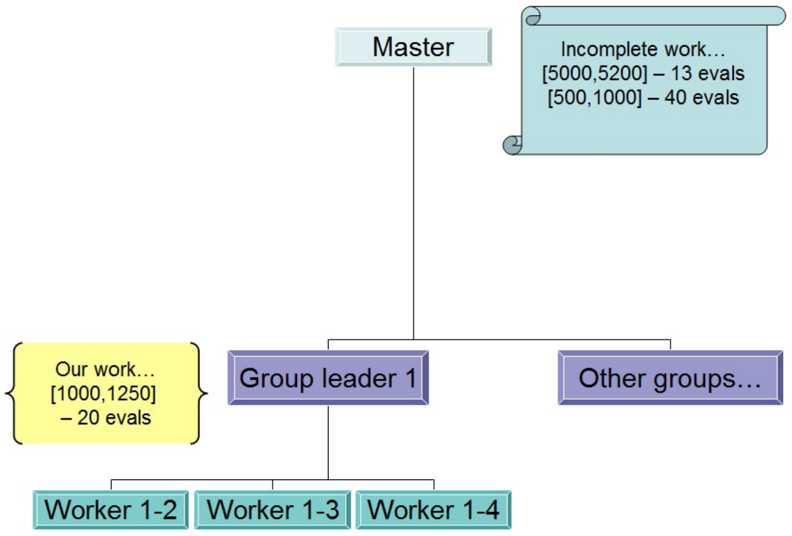


(a) The master sends an interval from its list of incomplete work to group leader 1.



(b) Group leader 1 relays this message to the workers in its group, since the master never directly communicated with them.

Figure 6.8.: A demonstration of TraceMin's dynamic load balancing (continued)



(a) Group 1 runs TraceMin on the interval (1000,1250).

Figure 6.8.: A demonstration of TraceMin's dynamic load balancing (continued)

7 COMPETING EIGENSOLVERS

We compare TraceMIN with several state of the art packages for computing eigenpairs of sparse symmetric eigenvalue problems such as

- SLEPc: an eigensolver package built on top of Argonne National Laboratory's PETSc, which implements a variety of different eigensolvers [22–26]
- Anasazi: the eigensolver package of Sandia National Laboratory's Trilinos library [20,36]
- FEAST: Eric Polizzi's contour integration eigensolver package [37,38]

The methods included in these packages are described in this section.

7.1 Arnoldi, Lanczos, and Krylov-Schur

These three methods are very similar to the power iteration for computing the largest eigenpair of a matrix, except that the power iteration uses a constant subspace dimension (like TraceMin) and these methods use expanding subspaces (like TraceMin-Davidson). The basic Arnoldi iteration generates a Krylov subspace of A one vector at a time as in algorithm 7, then computes the eigenpairs of $V^TAVX = X\Theta$, where V is the basis of that Krylov subspace[1]. The vectors $Y = VX$ are approximate eigenvectors of A, and the diagonal entries of Θ are the approximate eigenvalues. When the subspace becomes too large, we restart, keeping the most important vectors of the subspace and discarding the rest (just like TraceMin-Davidson). We can add one vector to the subspace at each iteration, or many if we're using block-Arnoldi.

[1]In the Arnoldi iteration, V^TAV is upper Hessenberg. If A is symmetric, V^TAV is tridiagonal and we have the Lanczos iteration.

Note that this method cannot compute the eigenpairs of a generalized eigenvalue problem without a spectral transformation (i.e. $B^{-1}Ax = \lambda x$).

If we seek the smallest eigenpairs of a matrix, we generally work with A^{-1} rather than A, which is referred to as shift-and-invert mode. Note that we would never form the matrix A^{-1} explicitly; at each iteration, we must solve a linear system $Av_k = v_{k-1}$. We may use either a direct or preconditioned iterative method, but the solution must be accurate. In general, if we want the relative residual of our eigenvalues to be less than 10^{-q}, these linear systems should be solved with a relative residual no more than 10^{-q-1}.

Algorithm 7 Arnoldi iteration

```
Require: $A \in \mathbb{R}^{n \times n}$
    $v_0 \in \mathbb{R}^{n \times 1}$
1: $v_1 = \frac{v_0}{\|v_0\|_2}$
2: for $k = 2 \to$ maxit do
3:     $v_k = Av_{k-1}$
4:     for $j = 1 \to k-1$ do
5:         $v_k = v_k - v_j v_j^T v_k$
6:     end for
7:     $v_k = \frac{v_k}{\|v_k\|_2}$
8: end for
```

Krylov-Schur is very similar to Arnoldi apart from how restarting is handled [39, 40]. When Arnoldi is restarted with a set of vectors V_0, it expects $V_0^T A V_0$ to still be upper-Hessenberg. Krylov-Schur relaxes the definition of an Arnoldi decomposition to avoid the difficulties Arnoldi has with deflation and restarting. Krylov Schur is implemented in the eigensolver package SLEPc, and a block form exists in Anasazi. Both forms are capable of using shift-and-invert mode, so we chose to run SLEPc's Krylov-Schur with shift-and-invert and Anasazi's block Krylov-Schur without shift-and-invert[2].

[2]In the absence of any spectral transformations, the only difference between using Krylov-Schur to compute the smallest eigenpairs or the largest is which vectors are kept upon restart.

7.1.1 Krylov-Schur with multisectioning

SLEPc's Krylov-Schur implementation is capable of multisectioning, but it must process each subinterval sequentially. All processes call the sparse factorization package MUMPS to factor the matrix in parallel [33, 34]. Because the MPI processes can not work independently, there is the potential for an overwhelming amount of communication, and Krylov-Schur will scale as MUMPS scales. Note that in our TraceMin-multisectioning implemention, MUMPS is never called by all processes simultaneously.

7.2 Locally Optimal Block Preconditioned Conjugate Gradient

The Locally Optimal Preconditioned Conjugate Gradient method minimizes (or maximizes) the generalized Rayleigh quotient at each iteration using a three term recurrence e.g.

$$y_{i+1} = \arg \min_{y \in \text{span}\{y_i, z_i, y_{i-1}\}} \rho(y) = \frac{y^T A y}{y^T B y}$$

where $z = M^{-1}r$ is the preconditioned residual [41]. The solution to this minimization problem is obtained via the Rayleigh-Ritz procedure, as outlined in algorithm 8. Although this algorithm demonstrates the single vector case, one can choose to use blocks instead, obtaining the Locally Optimal Block Preconditioned Conjugate Gradient method (LOBPCG).

This method may experience trouble in the B-orthonormalization step if the iterations stagnate, because in that case $y_i \approx y_{i-1}$. It is also only capable of finding extreme eigenpairs. We will compare against both the Trilinos implementation and SLEPC's interface to BLOPEX, which is Andrew Knyazev's own implementation [42].

7.3 Jacobi-Davidson

Jacobi-Davidson is an eigensolver which deals with the same constrained minimization problem as TraceMin, using the same projected-Krylov method. It was published

Algorithm 8 Locally Optimal Preconditioned Conjugate Gradient

```
Require: $A, B \in \mathbb{R}^{n\times n}$, both symmetric positive definite
    $v_1 \in \mathbb{R}^{n\times 1}$
 1: for $k = 1 \to$ maxit do
 2:     $y_k = \frac{v_k}{\|v_k\|_B}$
 3:     $\theta_k = y_k^T A y_k$
 4:     $r_k = Ay_k - \theta_k B y_k$
 5:     $z_k = M^{-1} r_k$
 6:     if $k > 1$ then
 7:         $V_k = [y_{k-1}\ y_k\ z_k]$
 8:     else
 9:         $V_k = [y_k\ z_k]$
10:     end if
11:     $B$-orthonormalize $V_k$
12:     $H = V_k^T A V_k$
13:     Solve the small dense eigenvalue problem $HX = X\Theta$
14:     $v_{k+1} = V_k x_1$, where $x_1$ is the eigenvector corresponding to the smallest eigen-
        value of $H$
15: end for
```

in 1996, 14 years after the TraceMin concept was first published by Ahmed Sameh and John Wisniewski [9, 43]. In their 1996 publication, Sleijpen and van der Vorst, applied it to the nonsymmetric case without a proof of global convergence. Later, they popularized their scheme for the symmetric case for which TraceMIN proved convergence much earlier. Unlike TraceMin, Jacobi-Davidson extracts its Ritz vectors from a subspace that expands at each iteration; this expanding subspace concept was later incorporated into the trace-minimization algorithm and given the name TraceMin-Davidson in 2000 [17]. TraceMin and TraceMin-Davidson use a very conservative method to compute their shifts, whereas Jacobi-Davidson chooses the shifts to be equal to the Ritz values. These Ritz values are frequently gross overestimates for the true eigenvalues of a matrix and can result in very slow convergence, or convergence to the wrong set of eigenpairs entirely. We will compare our eigensolver with the SLEPc implementation of Jacobi-Davidson. In order to avoid convergence issues caused by the original Jacobi-Davidson shifting strategy, the SLEPc developers chose to avoid shifting until the residual becomes very small.

7.4 Riemannian Trust Region method

The Riemannian Trust Region (RTR) method is very similar to TraceMin in that it also seeks to minimize the function

$$\hat{f}_Y(\Delta) = \operatorname{trace}\left(\left((Y-\Delta)^T B (Y-\Delta)\right)^{-1}\left((Y-\Delta)^T A (Y-\Delta)\right)\right) \tag{7.1}$$

for all $\Delta \perp_B Y$ [44, 45]. Assuming Y has been B-orthonormalized, the Taylor series expansion of $\hat{f}_Y$ about $\Delta = 0$ yields the following model, which is used by RTR

$$m_Y^{\text{RTR}}(\Delta) = \operatorname{trace}\left(Y^T A Y\right) - \operatorname{trace}\left(2Y^T A\Delta\right) + \frac{1}{2}\operatorname{trace}\left(2\Delta^T\left(A\Delta - B\Delta Y^T A Y\right)\right) \tag{7.2}$$

TraceMin approximates the Hessian of the matrix as $2A$, giving us

$$m_Y^{\mathrm{TM}}(\Delta) = \operatorname{trace}\left(Y^T A Y\right) - \operatorname{trace}\left(2Y^T A\Delta\right) + \frac{1}{2}\operatorname{trace}\left(2\Delta^T A\Delta\right) \tag{7.3}$$

RTR's model is more accurate and provides a better (superlinear) convergence rate, but the individual TraceMin iterations are cheaper.

In the absence of shifts, TraceMin tends to reduce the trace very quickly in its first few iterations before the trace levels off; RTR does the opposite, reducing the trace very slowly over the first few iterations due to the trust region constraint. Our tests will show comparisons with Chris Baker's RTR implementation in Trilinos [44].

7.5 FEAST

FEAST is Eric Polizzi's eigensolver package, which was recently adopted into the Intel Math Kernel Library. This eigensolver works by performing a contour integration at each iteration [37, 38]. As a result, FEAST treats all matrices as complex and requires considerably more storage than TraceMin. It must solve many linear systems $(Z_j B - A) Q_j = V$ at each iteration, one for each contour point. Since all input matrices are treated as complex, it is difficult to use iterative methods to solve the systems. In our comparisons, we use FEAST v 2.1 with its default linear solver, PARDISO [31,32].

Another result of the contour integration is that FEAST requires a lot of information from the user. It needs to know both the interval containing all eigenvalues of interest, as well as an accurate estimate of the number of eigenvalues within that interval. Although FEAST is one of the few eigensolver packages currently capable of multisectioning, it does require the user to explicitly provide the subintervals; it does not determine them on its own like TraceMIN does.

Algorithm 9 FEAST

```
Require: $A, B \in \mathbb{R}^{n\times n}$, $A$ symmetric and $B$ symmetric positive definite
    subspace dimension $s$
    number of Gaussian quadrature points $N_e$
    Gauss nodes $n_e$, $1 \leq j \leq N_e$ weights $\omega_j$, $1 \leq j \leq N_e$
    desired interval $[\lambda_{\min}, \lambda_{\max}]$
    $V \in \mathbb{R}^{n\times s}$
 1: for $k = 1 \rightarrow$ maxit do
 2:   Set $Q = 0$, $Q \in \mathbb{R}^{n\times s}$
 3:   Set $\sigma = (\lambda_{\max} - \lambda_{\min})/2$
 4:   for $j = 1 \rightarrow N_e$ do
 5:     Compute $\theta_j = -(\pi/2)(n_j - 1)$
 6:     Compute $Z_j = (\lambda_{\max} + \lambda_{\min})/2 + \sigma e^{i\theta_j}$
 7:     Solve $(Z_j B - A) Q_j = V$
 8:     Compute $Q = Q - (\omega_j/2) \Re\left\{\sigma e^{i\theta_j} Q_j\right\}$
 9:   end for
10:   Form $A_Q = Q^T A Q$ and $B_Q = Q^T B Q$
11:   Solve the eigenvalue problem $A_Q X = B_Q X \Sigma$
12:   Compute the Ritz vectors $Y = QX$
13:   Check convergence
14:   $V = BY$
15: end for
```

8 NUMERICAL EXPERIMENTS

8.1 Target hardware

The TraceMIN algorithm can be implemented on any parallel computing platform. This software implementation is aimed at the following broad class of parallel architectures. We assume a distributed memory system consisting of a large number of compute nodes that are interconnected via a high performance network, where each node consists of several cores. Our results were obtained on the following architectures:

- a Linux cluster with multicore nodes. Each node has two 12-core Intel Xeon E5-2697 v2 processors running at 2.7 GHz, with 64 GB of memory per node. These nodes are also interconnected via a fast Infiniband switch. We will refer to this architecture as *endeavor -1*. All programs were run with either 12 threads or 12 MPI processes per node on this architecture.

- a Linux cluster with multicore nodes. Each node has two 14-core Intel processors running at 2.6 GHz, with 64 GB of memory per node. These nodes are also interconnected via a fast Infiniband switch. We will refer to this architecture as *endeavor -2*. All programs were run with either 14 threads or 14 MPI processes per node on this architecture.

Since the Trilinos team is still working on improving the performance of their code with OpenMP, we ran all Trilinos code (including our Trilinos-based implementations of TraceMin and TraceMin-Davidson) with multiple processes per node. Similarly, we ran the SLEPc tests with pure MPI. Both FEAST and our Fortran implementations of TraceMin-Sampling and TraceMin-Multisectioning were run with multiple threads per node.

8.2 Computing a small number of eigenpairs

In each case, we consider the desired eigenpairs to be converged if the relative residual satisfies the following criteria

$$\frac{\|r_i\|_2}{\sigma_i} < 10^{-5}$$

where σ_i is the i-th Ritz value.

The experiments are conducted both with and without preconditioning on endeavor -2. The preconditioner M is chosen such that

$$M^{-1} = \left(\left(I - A_0^{-1}A\right)^2 + \left(I - A_0^{-1}A\right) + I\right) A_0^{-1}$$

where A_0 is the diagonal matrix obtained via SPAI(0) [46]; we have essentially performed a small number of Richardson iterations.

Unless otherwise stated, each eigensolver used its default parameter values. TraceMin used a block size of $s = 2p$, where p is the number of desired eigenpairs. TraceMin-Davidson used a block size of $s = p$ and stores a maximum of 10 blocks in the subspace V. Upon restart, TraceMin-Davidson retains the $2p$ Ritz vectors corresponding to the smallest Ritz values and discards the rest. Both TraceMin and TraceMin-Davidson use projected-MINRES to solve the saddle point problem at each iteration if we do not use preconditioning. If we choose to take advantage of a preconditioner, we use block-diagonal preconditioned MINRES to solve the saddle point problems. Trilinos' block Krylov-Schur used a block size of $s = p$ and a maximum subspace dimension of $10p$. For SLEPc's Krylov-Schur with shift-and-invert, we chose to use MINRES as the inner linear solver with a tolerance of 10^{-6}.

The results are summarized in tables 8.1, 8.2, and 8.3.

Table 8.1: Robustness of various solvers on our test problems.
yes denotes that a solver succeeded on this problem on all numbers of MPI processes, and no means the solver failed for some reason

	Anasazi				SLEPc		
Matrix	**TD**	BKS	LOBPCG	RTR	KS	LOBPCG	JD
Poisson	yes	yes	no	yes	no	no	no
Flan_1565	yes	no	no	yes	no	no	yes
Hook_1498	yes	no	no	yes	no	no	yes
cage15	yes	yes	yes	yes	yes	yes	yes
nlpkkt240	yes	no	no	no	no	no	yes

Table 8.2: Running time ratios of various solvers on our test problems (without preconditioning)

	Anasazi				SLEPc		
Matrix	**TD**	BKS	LOBPCG	RTR	KS	LOBPCG	JD
Poisson	1.0	19.7	3.0	1.6	-	-	-
Flan_1565	1.0	-	-	1.3	-	-	2.1
Hook_1498	1.0	-	-	3.3	-	-	1.5
cage15	1.5	2.5	1.0	1.7	2.2	5.4	2.9
nlpkkt240	1.0	-	-	-	-	-	5.6

Table 8.3: Running time ratios of various solvers on our test problems (with preconditioning)

Matrix	**TD**	LOBPCG	RTR
Poisson	1.0	11.7	1.2
Flan_1565	1.0	1.2	1.8
Hook_1498	1.0	4.9	2.3
cage15	2.3	1.0	-

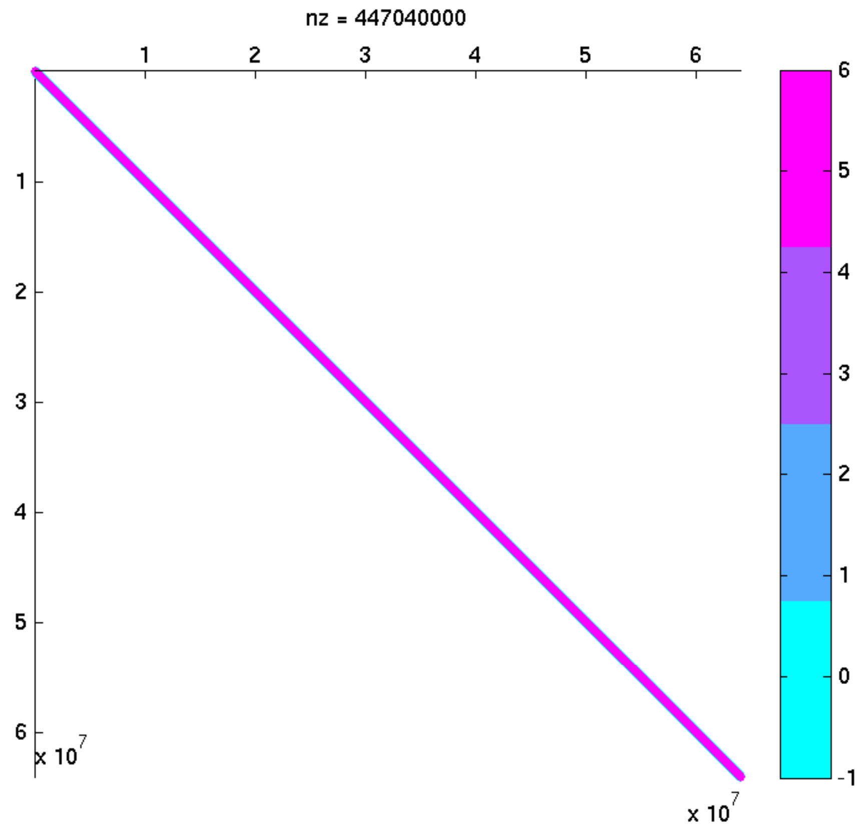


Figure 8.1.: Sparsity pattern of Laplace3D

8.2.1 Laplace3D

For this problem, A is the 3D discretization of the Laplace operator on a unit cube of order 64 million with roughly 450m nonzeros (Figure 8.1). This matrix is symmetric positive definite and diagonally dominant. We seek the four smallest eigenvalues

$$\left(1.84 \times 10^{-4}, 3.68 \times 10^{-4}, 3.68 \times 10^{-4}, 3.68 \times 10^{-4}\right)$$

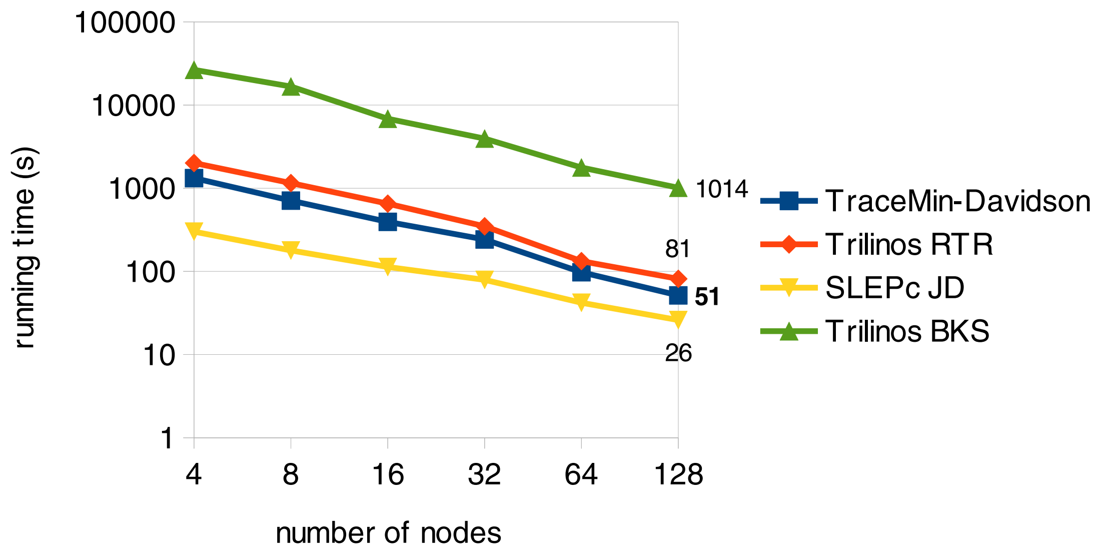
along with their associate eigenvectors. Note that one of these eigenvalues has a multiplicity of three.

In figure 8.2, it appears as though SLEPc's Jacobi-Davidson is the fastest method; it is roughly twice as fast as TraceMin-Davidson. However, since it uses a block size of 1, SLEPc's Jacobi-Davidson failed to capture the correct multiplicity of eigenvalue 3.68×10^{-4}. When we tried to increase the block size to 4, Jacobi-Davidson crashed. Again, Jacobi-Davidson and TraceMin-Davidson are very similar, so if we had been able to increase the block size, Jacobi-Davidson probably would have had comparable performance to our code. LOBPCG converged on 8, 16, and 32 nodes, but it crashed the rest of the time. Because we were not using a preconditioner, LOBPCG took a large number of iterations and stagnated, causing an orthogonalization error that resulted in termination of the program. SLEPc's Krylov-Schur implementation failed to converge in a reasonable amount of time because it took so long to solve the linear systems accurately (over 13 hours on 4 nodes and 2 hours on 128 nodes). TraceMin-Davidson was the fastest of the methods which found the correct eigenpairs, and it had a nearly optimal speed improvement up to 128 nodes.

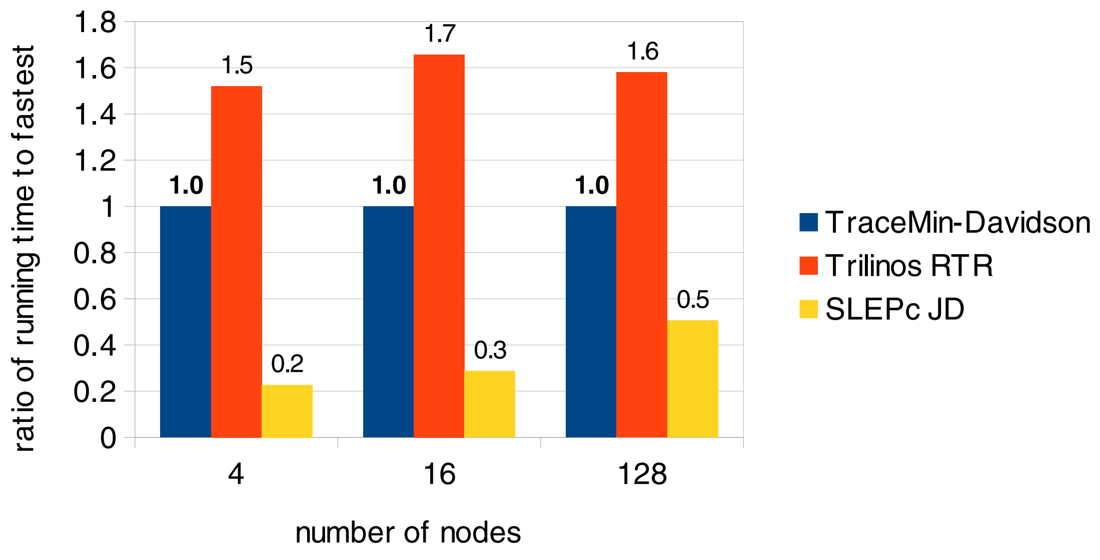
8.2.2 Flan_1565

Janna/Flan_1565 is a symmetric positive definite banded matrix in the Tim Davis collection [47], representing a 3D model of a steel flange (Figure 8.3). It is order 1.5 million, with approximately 100 million nonzero entries. We seek the four smallest eigenvalues and their associated eigenvectors.

Figure 8.4 shows that all methods scaled quite well up to 32 nodes, then began to level off a bit. This is not surprising, given the relatively small size of the matrix and the fact that we have not assured any kind of load balancing. TraceMin-Davidson was the fastest method, though the other two related methods (Jacobi-Davidson and the Riemannian Trust Region method) were also able to solve the problem in a reasonable amount of time. Both the Trilinos and SLEPc implementations of LOBPCG failed to solve this problem, presumably because the iterations stagnated and resulted in a loss of orthogonality. Trilinos block Krylov-Schur failed to solve the problem in a reasonable amount of time (over 25 hours on 2 nodes and 2 hours on 128 nodes), and



(a) Scalability comparison



(b) Ratio of running times

Figure 8.2.: A comparison of several methods of computing the four smallest eigenpairs of Laplace3D (without preconditioning)

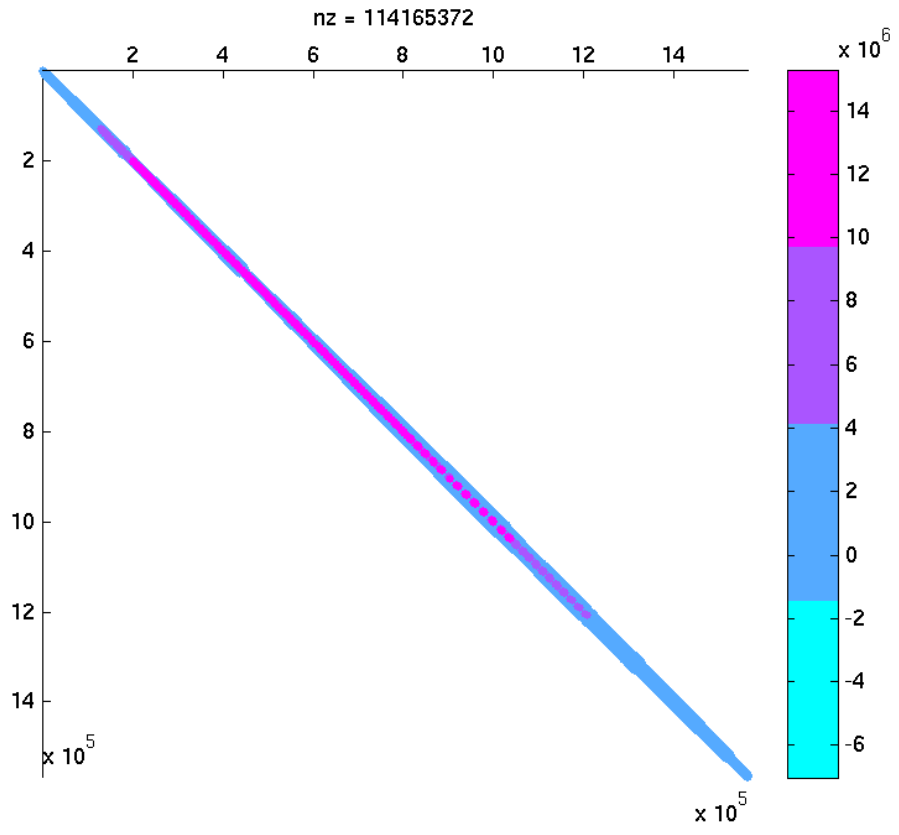
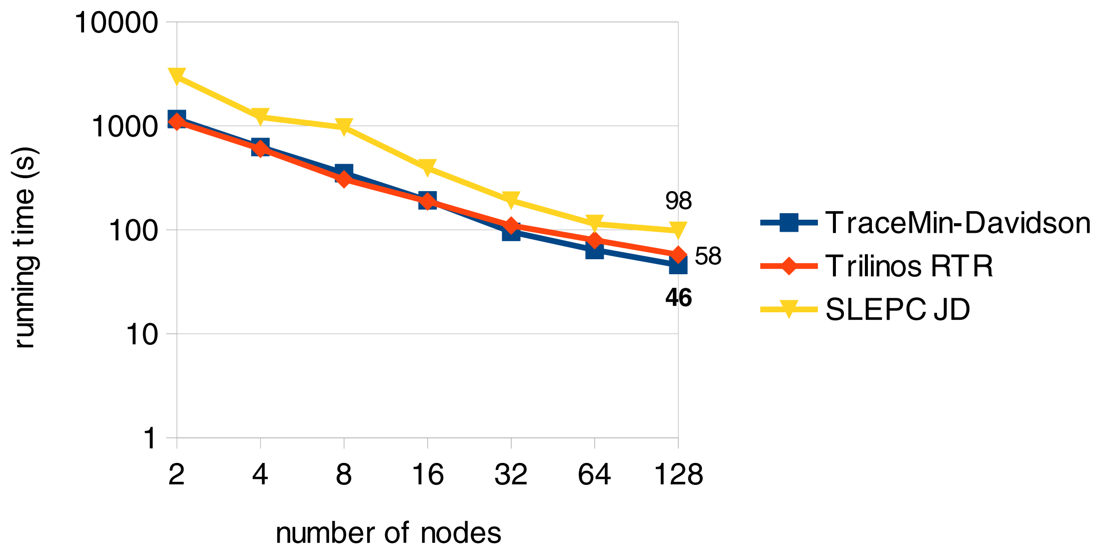


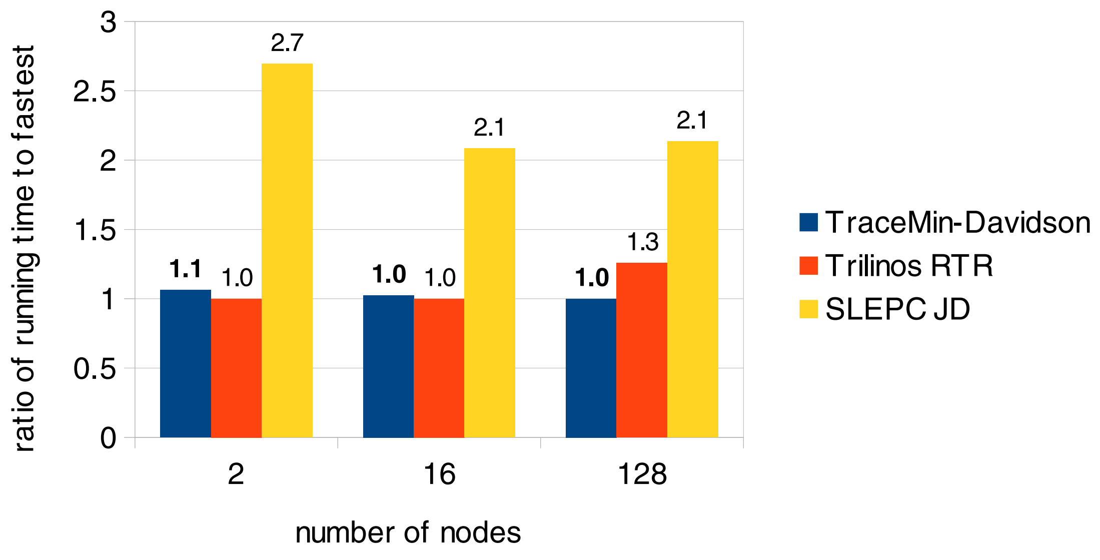
Figure 8.3.: Sparsity pattern of Flan_1565

SLEPc's Krylov-Schur with shift-and-invert failed to solve the linear system to the required degree of accuracy and terminated.

If we try the same test with preconditioning, we obtain the results illustrated in figure 8.5. Preconditioning took the running time of TraceMin-Davidson from 46s down to 27s on 128 nodes, if we use the block diagonal preconditioned MINRES previously described to solve the saddle point problem at each iteration. In fact, we note that using the block diagonal preconditioning in this case is over twice as fast as using projected MINRES. With preconditioning, the Trilinos implementation of LOBPCG was able to converge on 2, 4, and 16 nodes because the preconditioner caused the iterations to stagnate less frequently, but it still crashed most of the time.

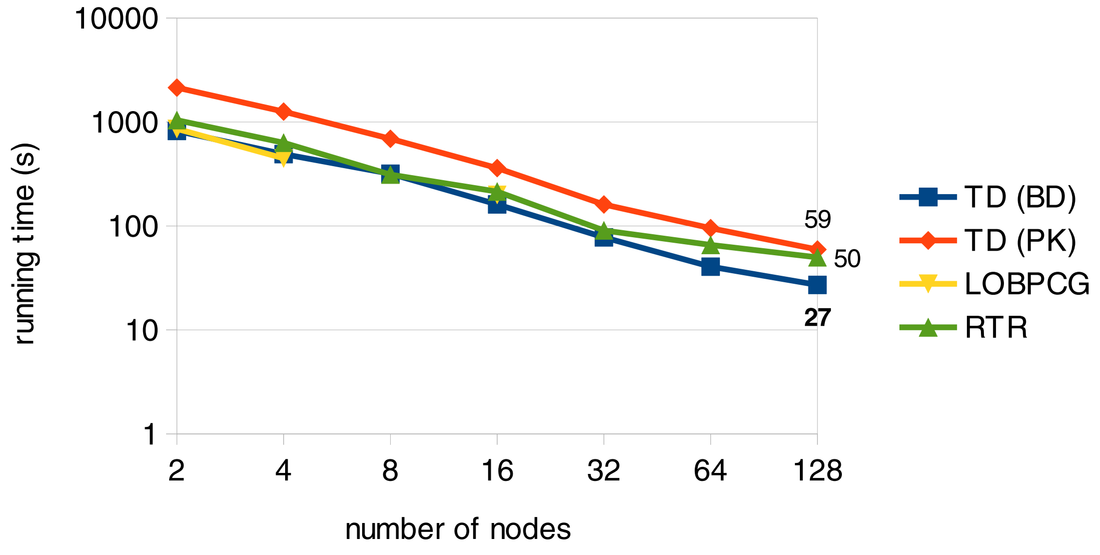


(a) Scalability comparison

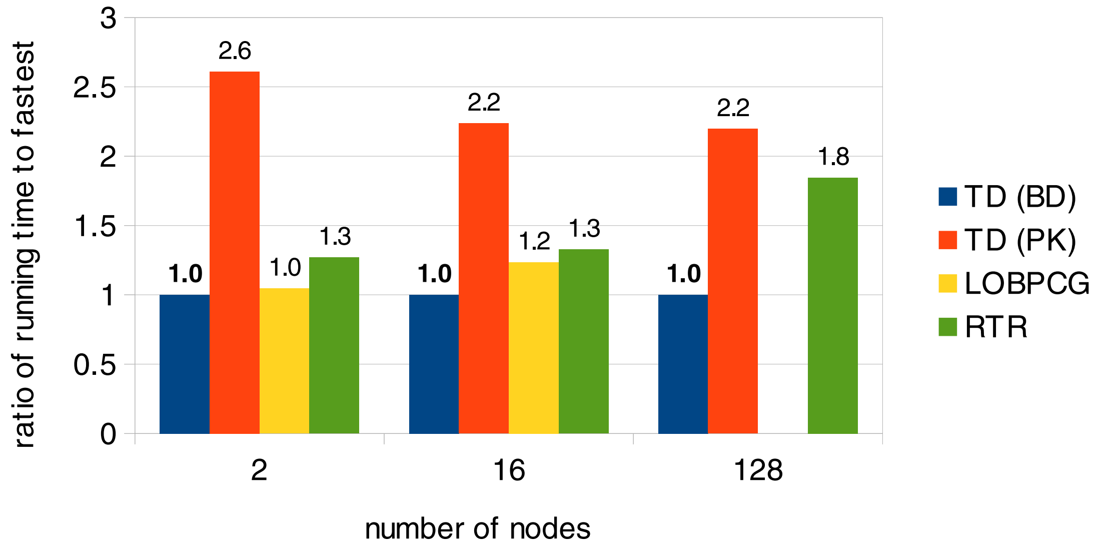


(b) Ratio of running times

Figure 8.4.: A comparison of several methods of computing the four smallest eigenpairs of Janna/Flan_1565 (without preconditioning)



(a) Scalability comparison



(b) Ratio of running times

Figure 8.5.: A comparison of several methods of computing the four smallest eigenpairs of Janna/Flan_1565 (with preconditioning)

8.2.3 Hook_1498

Janna/Hook_1498 is a symmetric positive definite banded matrix in the Tim Davis collection, representing a 3D model of a steel hook (Figure 8.6). It is order 1.5 million, with approximately 60 million nonzero entries. We seek the four smallest eigenvalues and their associated eigenvectors.

Figure 8.7 shows that Jacobi-Davidson was the fastest method up to 16 nodes (although TraceMin-Davidson was still competitive), but on larger numbers of nodes, Jacobi-Davidson fails to scale well[1]. This is likely due to the fact that SLEPc is incapable of using block or pseudo-block linear solvers. Because our TraceMin-Davidson implementation uses pseudo-block solvers, it continued to scale up to 64 nodes. Once again, both implementations of LOBPCG crashed due to orthogonalization errors, and Krylov-Schur could not solve the linear systems to a sufficient degree of precision in the shift-and-invert mode.

If we try the same test with preconditioning, we obtain the results of figure 8.8. Preconditioning did not greatly impact the running time of TraceMin-Davidson, but it did cause the Riemannian Trust Region method to converge a bit faster. It also prevented LOBPCG from stagnating.

8.2.4 cage15

For this example, we will be computing the Fiedler vector of a directed weighted graph with one strongly connected component [2], vanHeukelum/cage15 from the Tim Davis collection; we will refer to this graph as G. G has approximately five million rows and one hundred million nonzeros. Recall that the weighted graph Laplacian A will be symmetric positive semi-definite with a null space of dimension 1. The null vector is the scaled vector of all 1s, which we provided to all eigensolvers. We wish

[1]BKS was over 30 times slower than TraceMin-Davidson. It is not competitive for Hook_1498 and has been excluded from figure 8.7b.

[2]In general, we would determine the strongly connected components using a Dulmage-Mendelsohn permutation and treat each one as a separate problem.

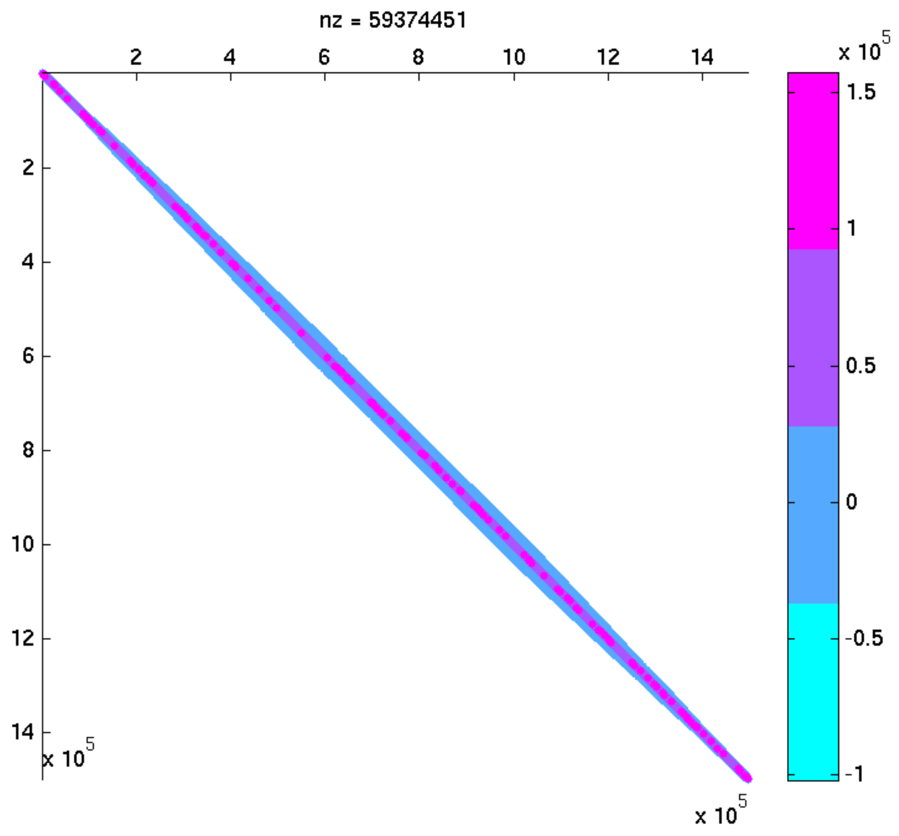
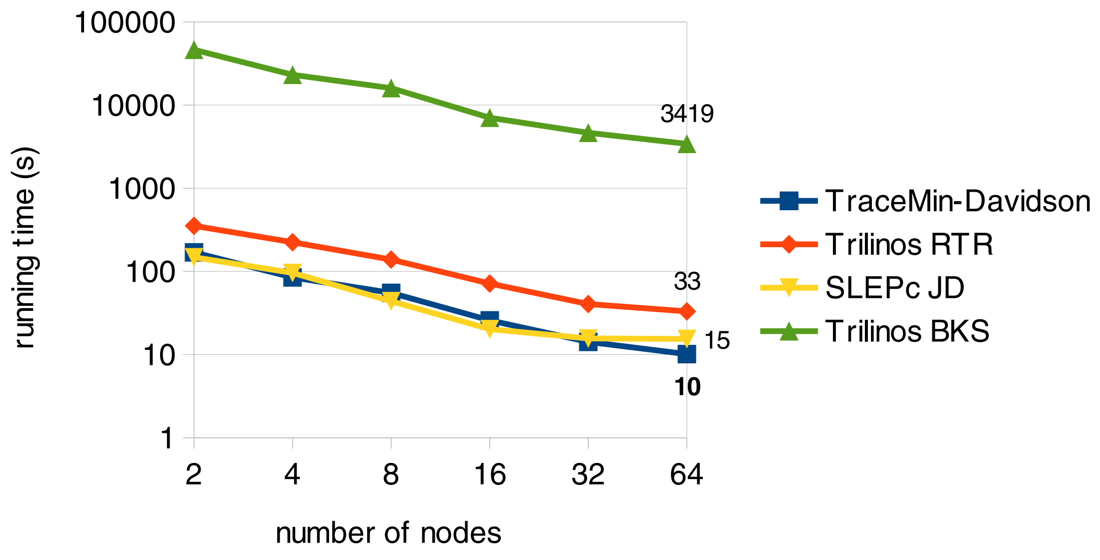
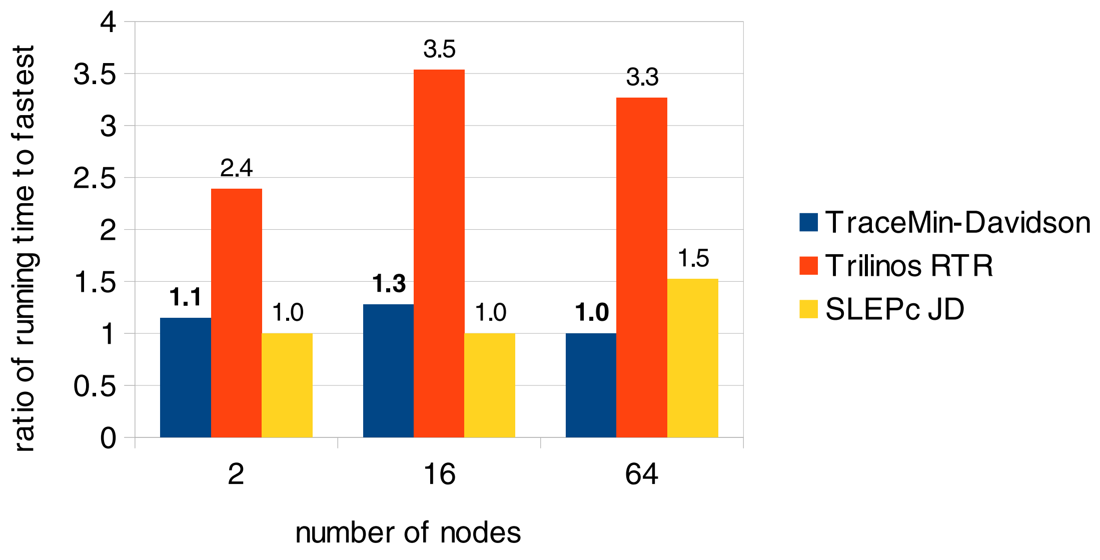


Figure 8.6.: Sparsity pattern of Hook_1498

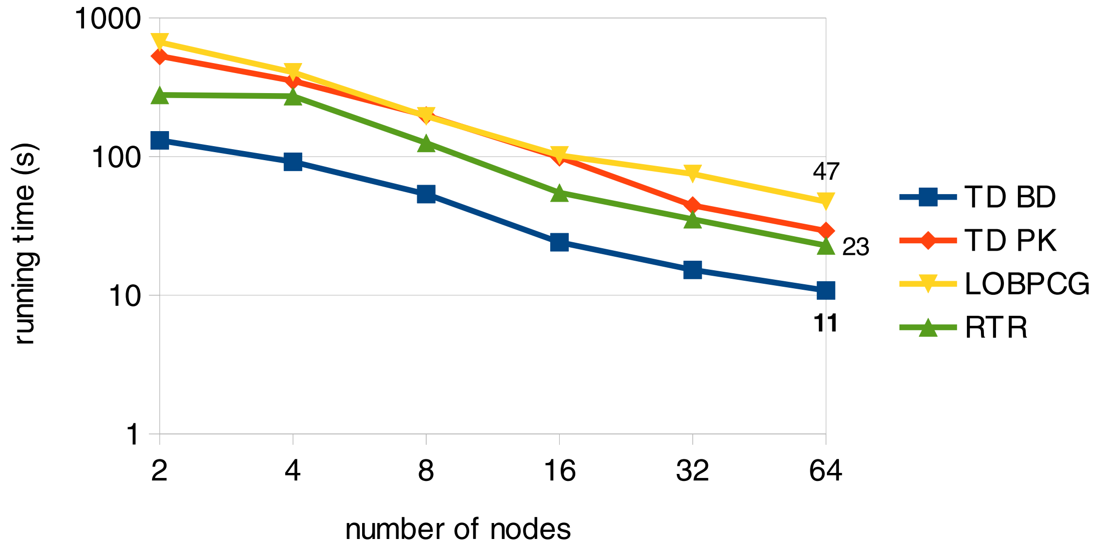


(a) Scalability comparison

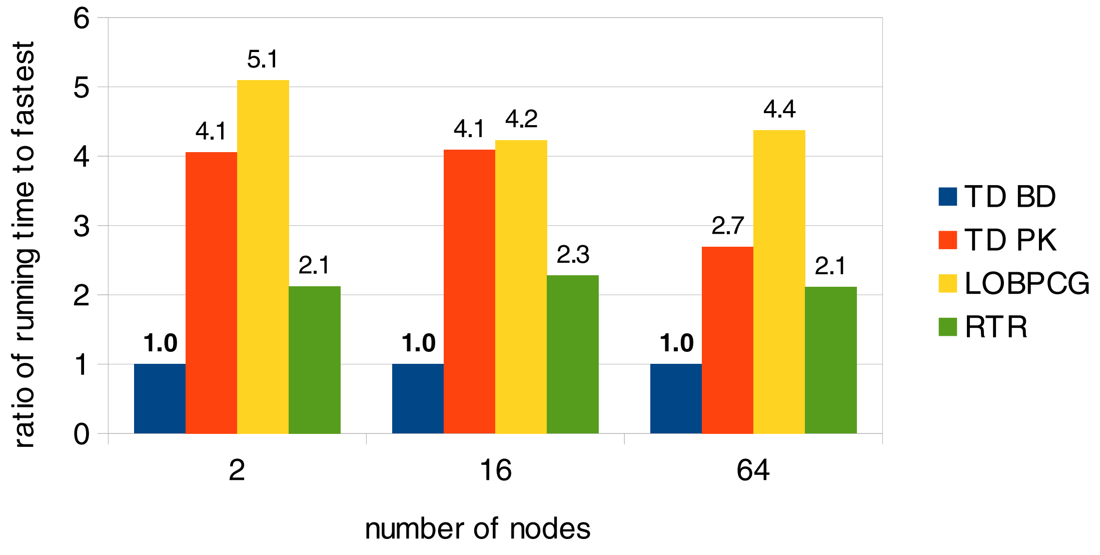


(b) Ratio of running times

Figure 8.7.: A comparison of several methods of computing the four smallest eigenpairs of Janna/Hook_1498 (without preconditioning)



(a) Scalability comparison



(b) Ratio of running times

Figure 8.8.: A comparison of several methods of computing the four smallest eigenpairs of Janna/Hook_1498 (with preconditioning)

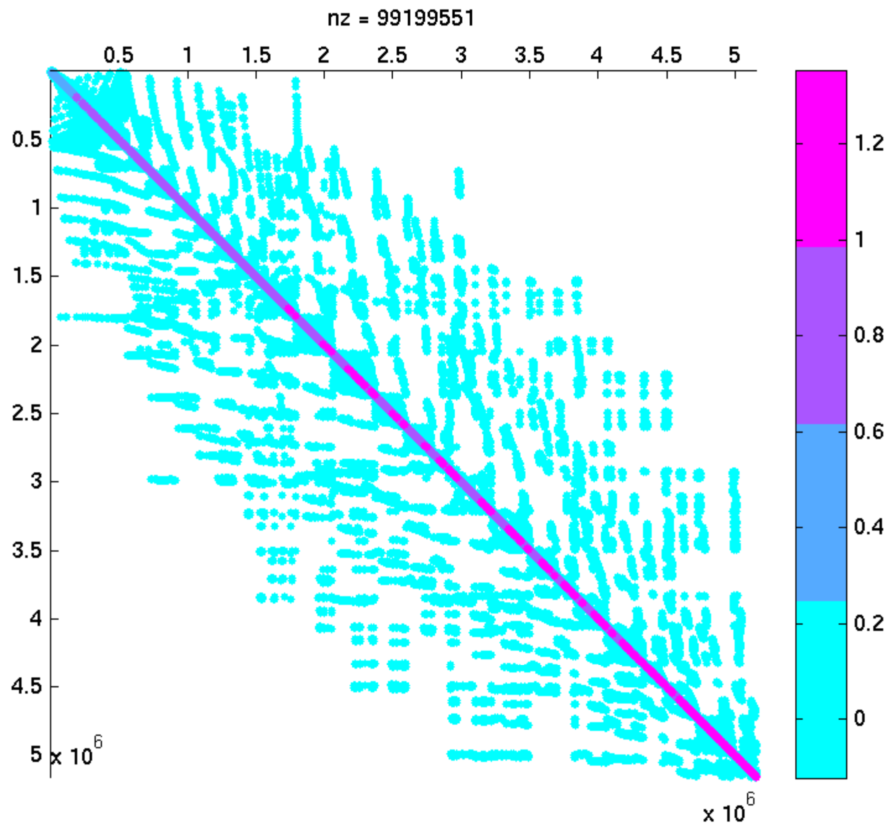


Figure 8.9.: Sparsity pattern of cage15

to find the smallest nonzero eigenvalue (the graph connectivity) and the associated eigenvector, which is referred to as the Fiedler vector. For this problem, we increased TraceMin's block size to $s = 6$ and set the maximum number of vectors to be stored in V to 20 for both TraceMin-Davidson and block Krylov-Schur.

Figure 8.10 shows a comparison between many different eigensolvers on this problem. We see that all methods performed very well because it was easy to solve linear systems involving A. The Trilinos implementation of LOBPCG was the fastest, but TraceMin-Davidson still performed quite well. The issue here was that the eigenvalues are clustered, and TraceMin-Davidson should have used a much weaker inner tolerance than it did, since the outer convergence rate was going to be poor regardless.

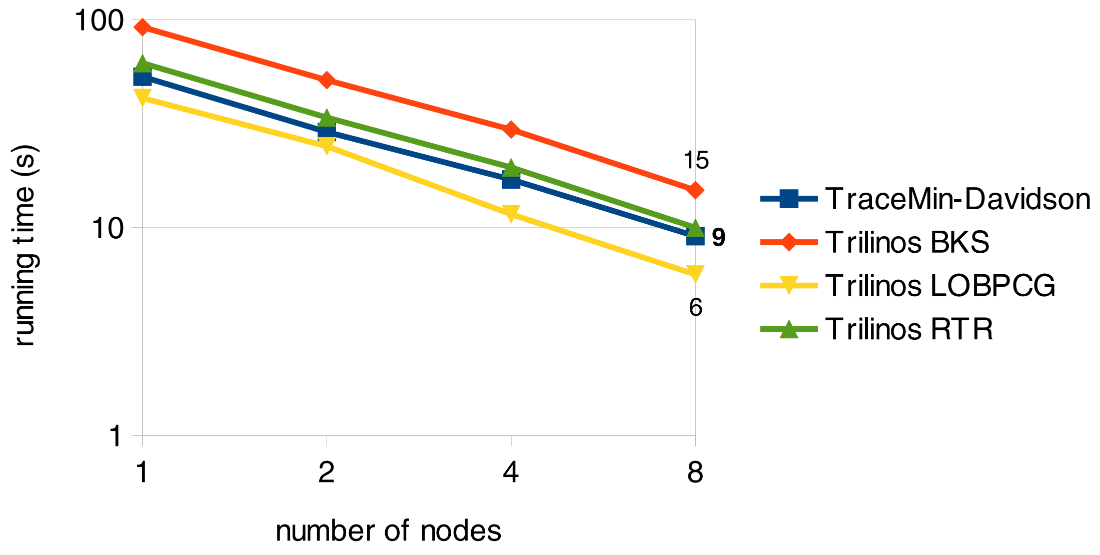
Figure 8.11 shows that we still did very well on this problem, in comparison with the other eigensolvers.

8.2.5 nlpkkt240

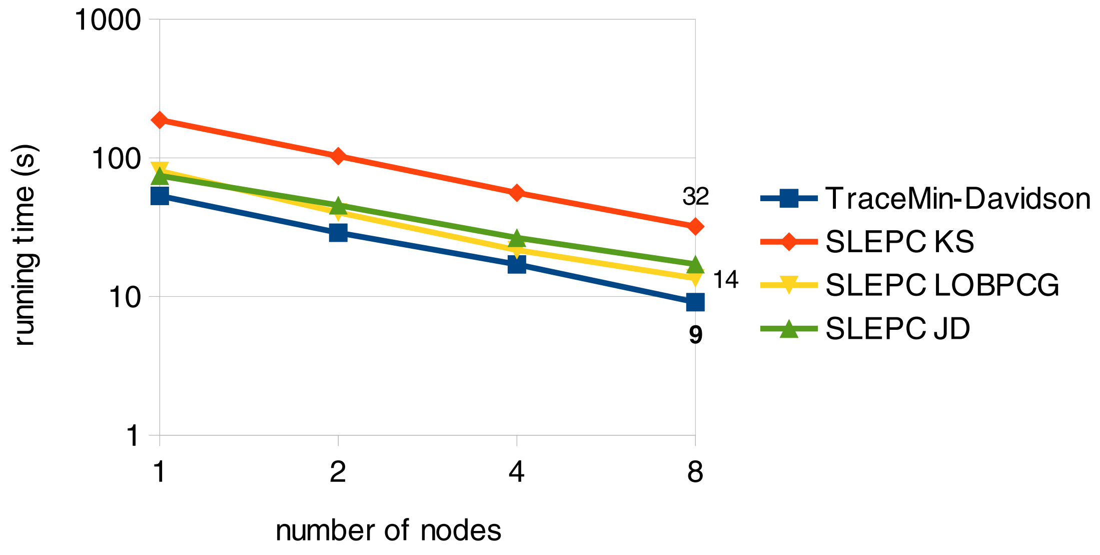
Schenk/nlpkkt240 is a symmetric indefinite KKT matrix in the Tim Davis collection of order 27 million with approximately 800 million nonzeroes. We reordered it to a banded matrix (figure 8.12), using symmetric reverse Cuthill-McKee, so that the matrix vector multiplications would be more efficient[3]. We seek the four smalest magnitude eigenvalues with their associated eigenvectors; note that these will be interior eigenpairs rather than extreme ones. The Riemannian Trust Region method and LOBPCG cannot compute interior eigenpairs without a spectral transformation such as spectrum folding.

Figure 8.13 shows the running time of both TraceMin-Davidson and SLEPc's Jacobi-Davidson implementation, both of which performed a harmonic Ritz extraction. Trilinos BKS took a prohibitively long time to converge (over 3 hours on 128 nodes), and SLEPc's Krylov-Schur with shift-and-invert failed to converge because the Krylov solver was unable to solve the linear systems to a sufficient degree of precision. LOBPCG (with specturm folding) also took too much time to be competitve. TraceMin-Davidson was over six times faster than Jacobi-Davidson, presumably because Jacobi-Davidson used more aggressive shifts which approximated eigenvalues that were much larger than the ones we desired. We also see that TraceMin-Davidson scaled almost perfectly, whereas Jacobi-Davidson did not scale as well on a large number of nodes. This is presumably due to the fact that TraceMin-Davidson used a pseudo-block solver, and SLEPc's Jacobi-Davidson did not.

[3]Technically, since both Trilinos and SLEPc allow the user to provide an operator rather than a matrix, we could have provided matvecs and linear solvers that took advantage of the special structure of this matrix.



(a) Comparison between TraceMin-Davidson and other Trilinos eigensolvers



(b) Comparison between TraceMin-Davidson and several SLEPc eigensolvers

Figure 8.10.: A comparison of several methods of computing the Fiedler vector for cage15

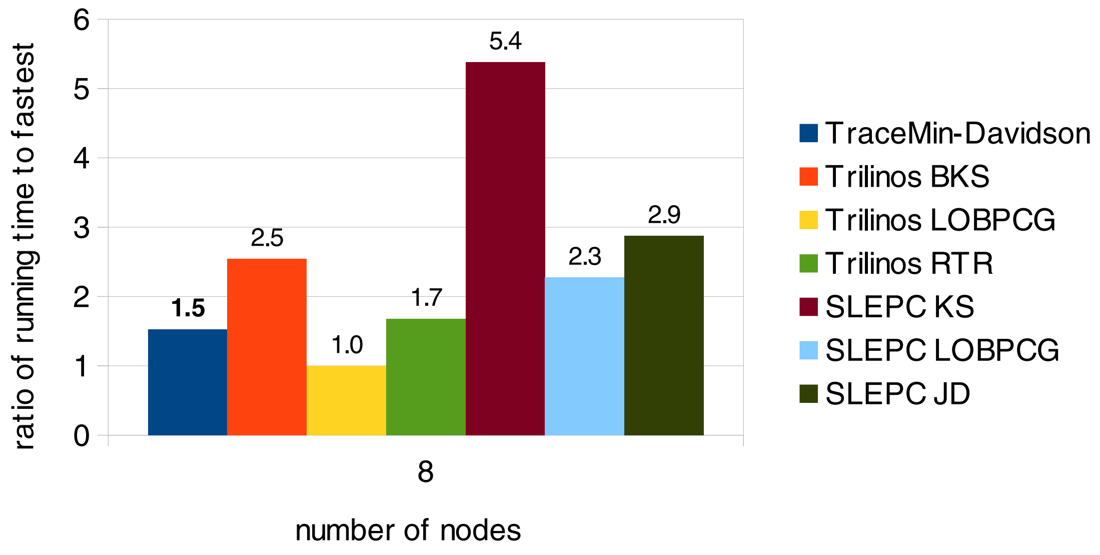


Figure 8.11.: Ratio of running times for computing the Fiedler vector of cage15

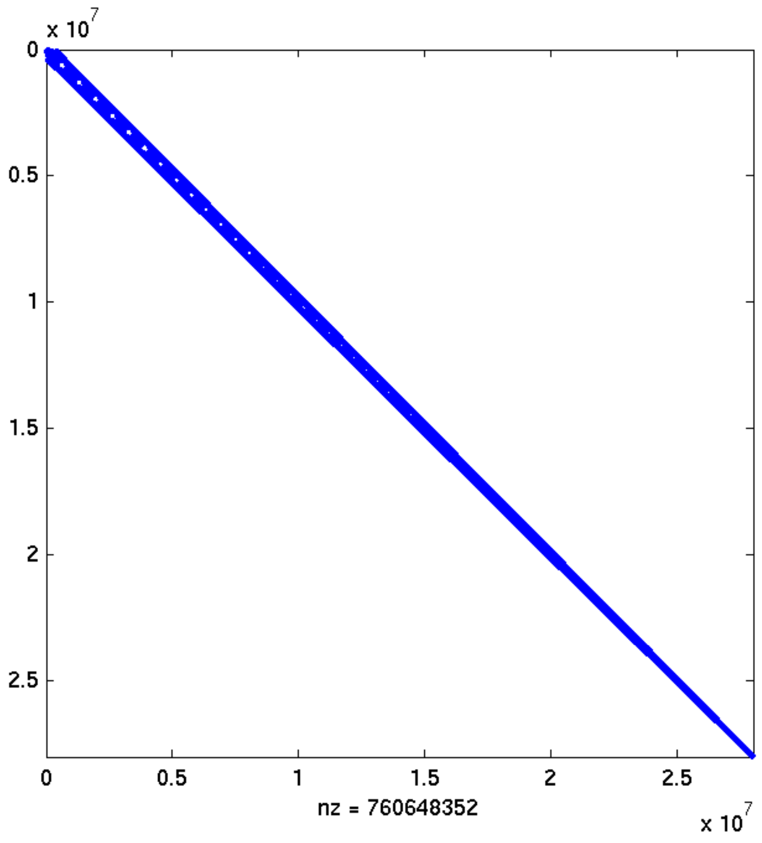
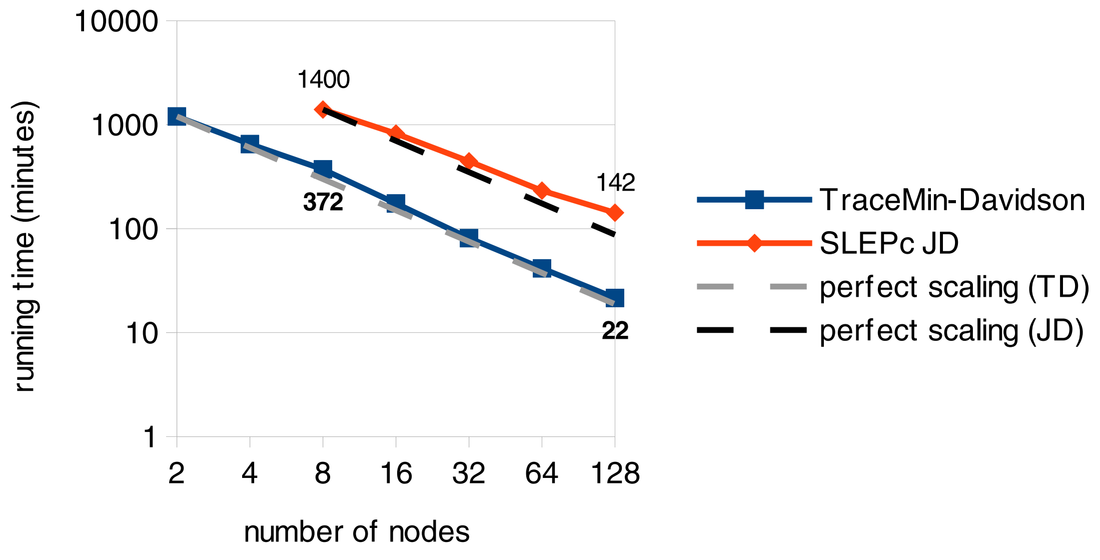
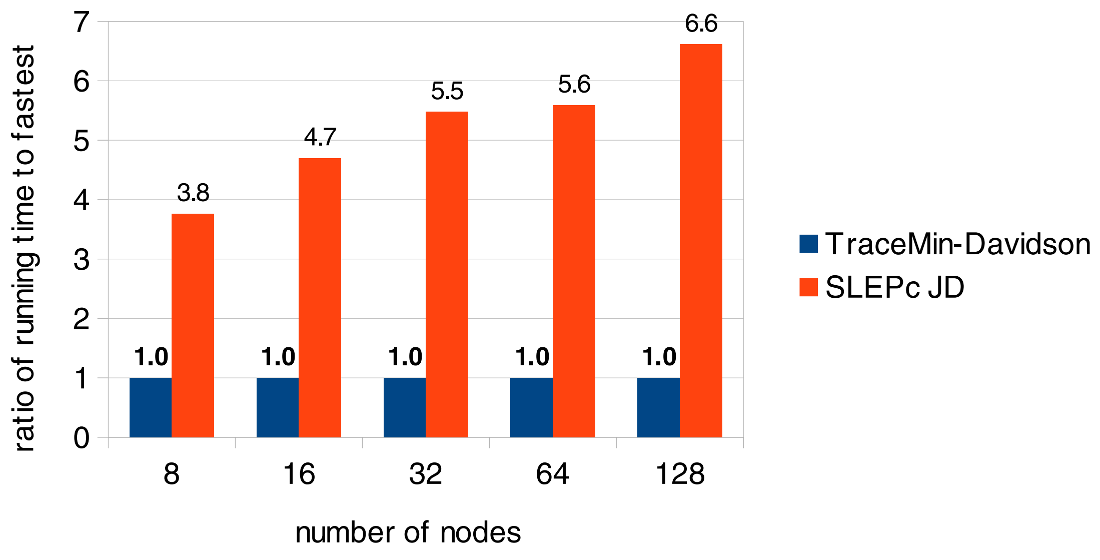


Figure 8.12.: Sparsity pattern of nlpkkt240 (after RCM reordering)



(a) Scalability comparison



(b) Ratio of running times

Figure 8.13.: A comparison of several methods of computing the four smallest eigenvalues of nlpkkt240

Table 8.4: Running time for TraceMin-Sampling on the Anderson problem

tolerance	running time (s)
10^{-5}	283
10^{-6}	306
10^{-9}	392

8.3 Sampling

The results in this section were obtained using a Fortran 90 implementation of TraceMin, using PARDISO to solve the linear systems arising at each iteration. We have converged when the relative residual

$$\frac{\|r_i\|_2}{\max\left(\|A\|_1, \|B\|_1\right)} < tol$$

These tests were run on endeavor -1.

8.3.1 Anderson model of localization

For this example, we will compute a few eigenpairs of an Anderson matrix of order one million closest to a set of shifts (pictured in figure 8.14). We will compute the four eigenvalues closest to 100 evenly spaced shifts in the interval $[-1, 1]$ using 100 MPI processes. Table 8.4 shows that it only took us five minutes to compute the 400 desired interior eigenpairs.

8.3.2 Nastran benchmark (order 1.5 million)

We will compute a few eigenpairs of the Nastran benchmark of order 1.5 million in this example. This is a generalized eigenvalue problem; the sparsity patterns of A and B are plotted in figure 8.15. We will compute the 4 eigenvalues closest to 100 evenly spaced shifts in the interval $[-0.01; 1,461,000]$ using 100 MPI processes.

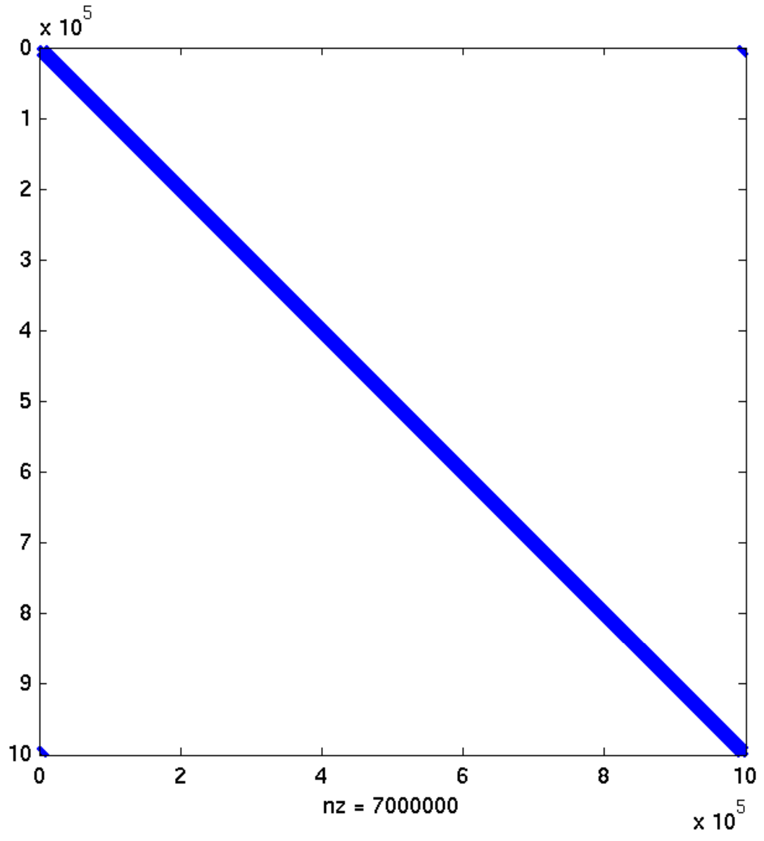


Figure 8.14.: Sparsity pattern of the Anderson matrix

Table 8.5: Running time for TraceMin-Sampling on the Nastran benchmark of order 1.5 million

tolerance	running time (s)
10^{-5}	21
10^{-6}	25
10^{-9}	40

Table 8.6: Running time for TraceMin-Sampling on the Nastran benchmark of order 7.2 million

tolerance	running time (s)
10^{-5}	181
10^{-6}	200
10^{-9}	302

Table 8.5 shows that it only took about 30 seconds to compute the 400 desired interior eigenpairs.

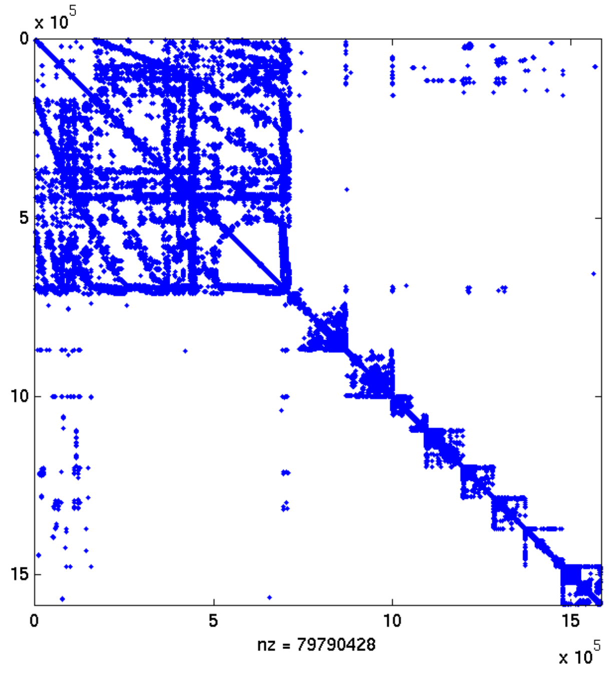
8.3.3 Nastran benchmark (order 7.2 million)

We will compute a few eigenpairs of the Nastran benchmark of order 7.2 million in this example. This is a generalized eigenvalue problem; the sparsity patterns of A and B are plotted in figure 8.16. We will compute the 4 eigenvalues closest to 100 evenly spaced shifts in the interval $[-0.01; 2,785,937.5]$ using 100 MPI processes. Table 8.6 shows that it only took a few minutes to compute the 400 desired interior eigenpairs.

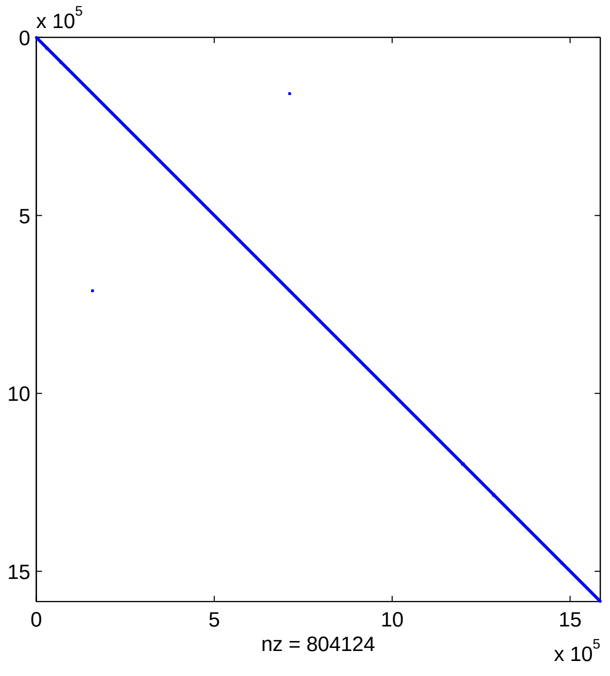
8.4 TraceMin-Multisectioning

In these examples, we seek all eigenpairs located in a large interval, with a relative residual

$$\frac{\|r_i\|_2}{\max\left(\|A\|_1, \|B\|_1\right)} < 10^{-6}$$

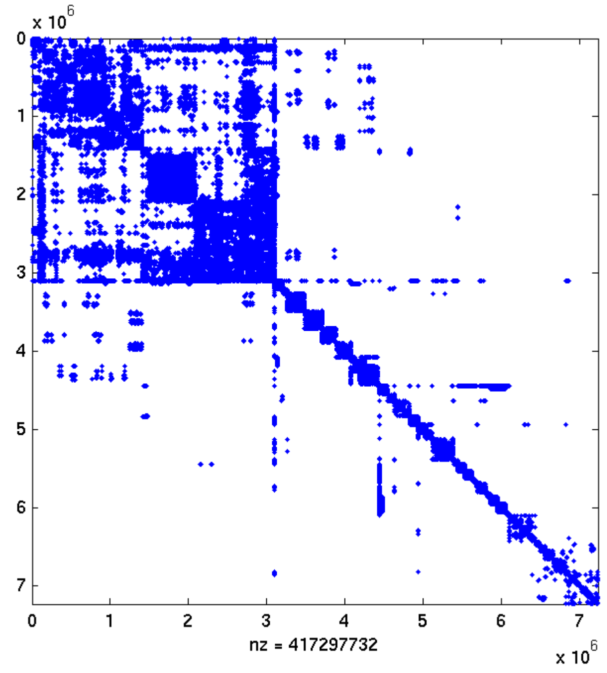


(a) Sparsity pattern of stiffness matrix A

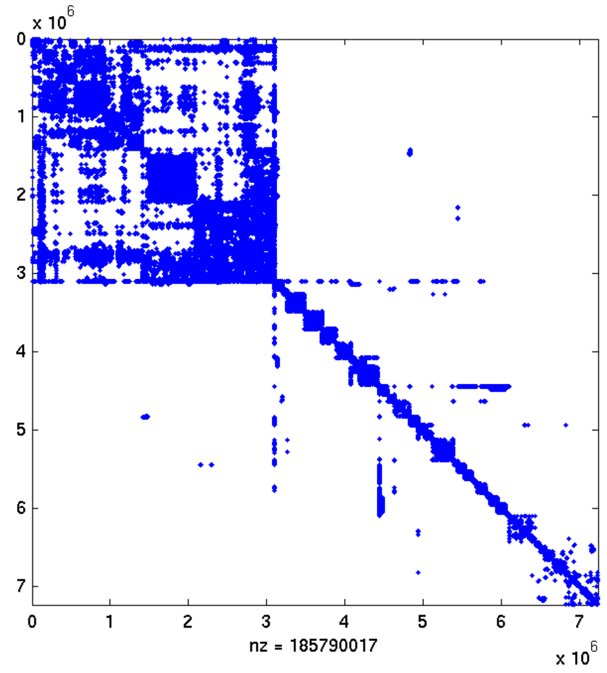


(b) Sparsity pattern of mass matrix B

Figure 8.15.: Sparsity patterns for the Nastran benchmark of order 1.5 million



(a) Sparsity pattern of stiffness matrix A



(b) Sparsity pattern of mass matrix B

Figure 8.16.: Sparsity patterns for the Nastran benchmark of order 7.2 million

Table 8.7: Running time comparison of FEAST and TraceMin-Multisectioning

Matrix	Interval	nev	**TraceMin**	FEAST	Speedup
Nastran 1.5m	[-0.01,1.461e6]	1000	59 s	121 s	2.2
Nastran 7.2m	[-0.01,2785937.5]	1000	418 s	-	-
Anderson	[-0.01,0.01]	1143	792 s	7910 s	10.0
af_shell10	[2000,2250]	1045	37 s	274 s	7.3
dielFilterV3real	[25,50]	2969	301 s	912 s	3.0
StocF-1465	[580,600]	4150	195 s	738 s	3.8

on the platform endeavor -2. This interval can contain thousands of eigenpairs, so we will be running the multisectioning code previously described. TraceMin-Multisectioning will subdivide the global interval until it has many smaller subintervals, each containing at most 20 eigenvalues. We will use 1 MPI process per node, with 14 threads per process, using PARDISO to compute the inertia and solve the linear systems.

We compared our code against both FEAST and SLEPc's Krylov-Schur with spectrum slicing. Since FEAST requires users to explicitly subdivide the interval themselves, we divided it into p equally sized pieces, where p is the number of MPI processes, and provided each MPI process with a single one of those pieces. We ran FEAST with 14 threads per MPI process, with 1 MPI process per node. SLEPc's Krylov-Schur spectrum slicing implementation does dynamically subdivide the interval, but the subintervals are dependent. As a result, they must be processed one at a time, using all MPI processes on a single group, which can result in poor scalability. No timing results are presented for SLEPc, since all tests failed with the error message: "PETSC ERROR: Unexpected error in Spectrum Slicing! Mismatch between number of values found and information from inertia." The results are summarized in table 8.7.

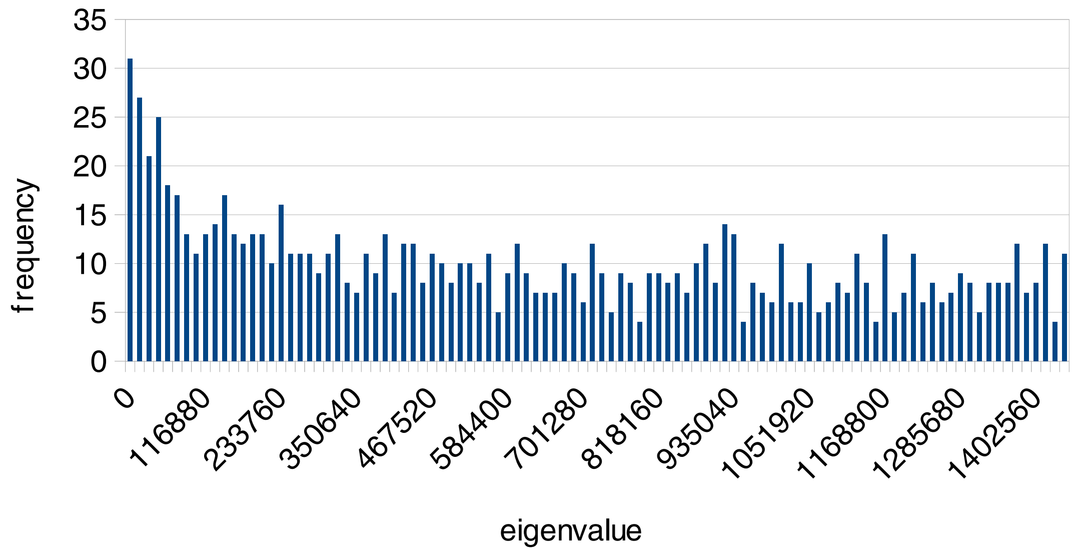
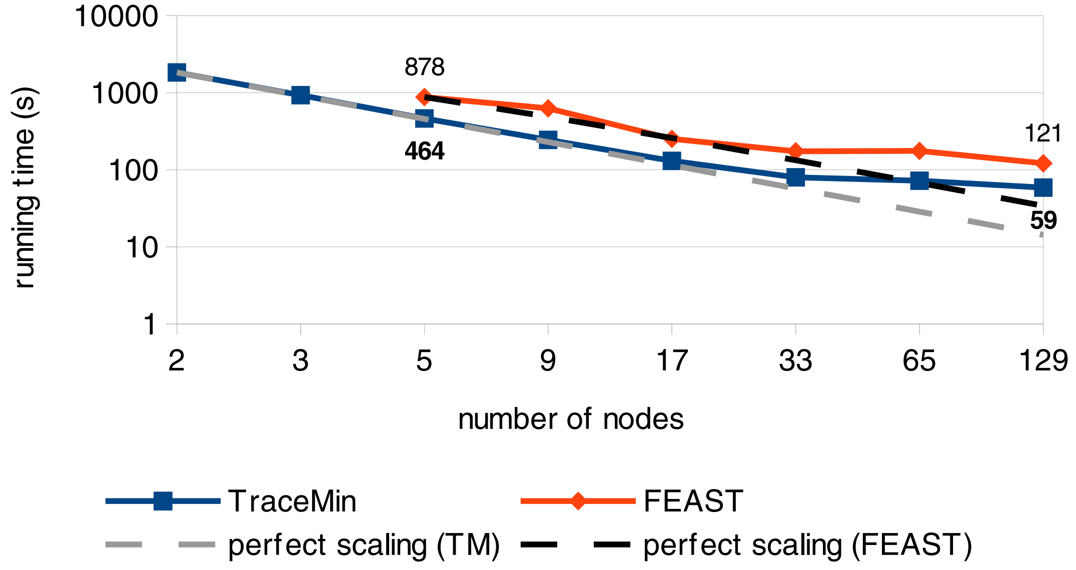


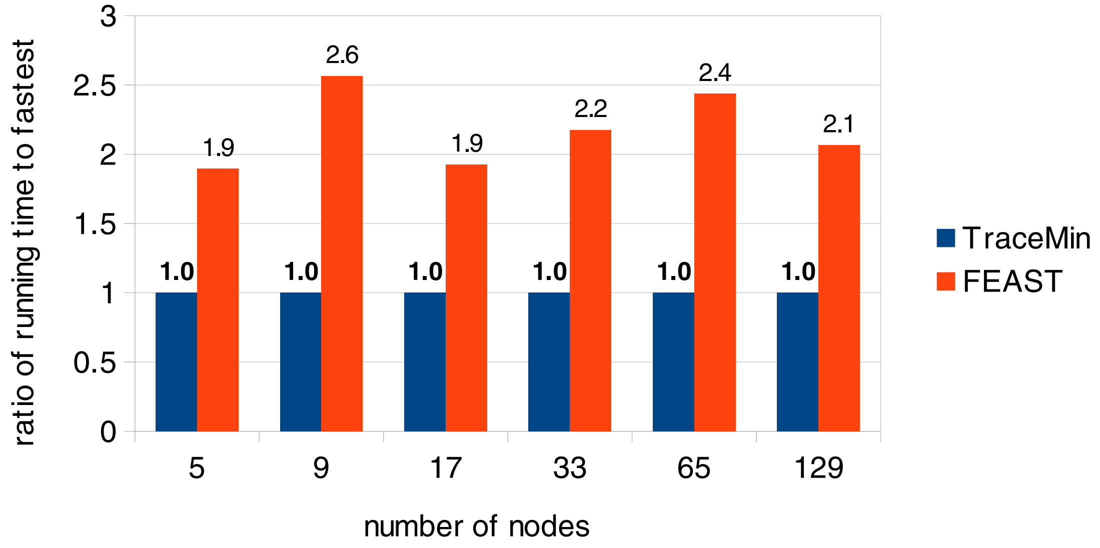
Figure 8.17.: Histogram of the eigenvalues of interest for the Nastran benchmark (order 1.5 million)

8.4.1 Nastran benchmark (order 1.5 million)

We seek all the eigenpairs in the region [-0.01,1.461e6], which contains 1000 eigenpairs (figure 8.17). TraceMin and FEAST both performed reasonably well on this problem up to 33 nodes (as shown in figure 8.18), then failed to continue scaling. However, TraceMin-Multisectioning still managed to compute all 1000 eigenpairs in roughly one minute on as few as 33 nodes, and it was over twice as fast as FEAST. We also see that the subdivision of intervals, which could be considered to be a preprocessing step, took less than 10% of the total running time. FEAST crashed on both 2 and 3 nodes because it could not store so many eigenvectors on a single node (along with the matrices A and B, and also the complex factorization).

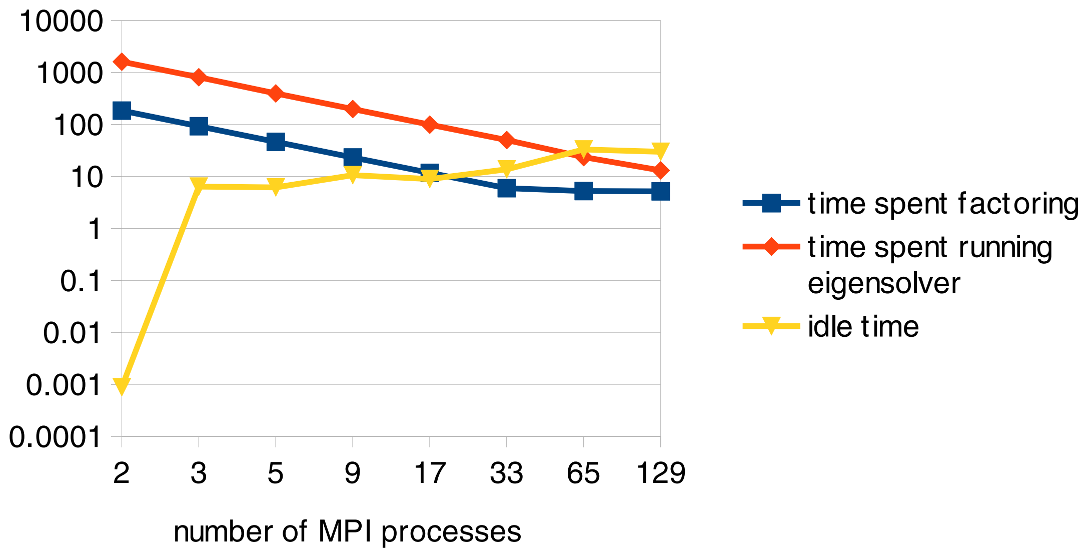


(a) Scalability comparison

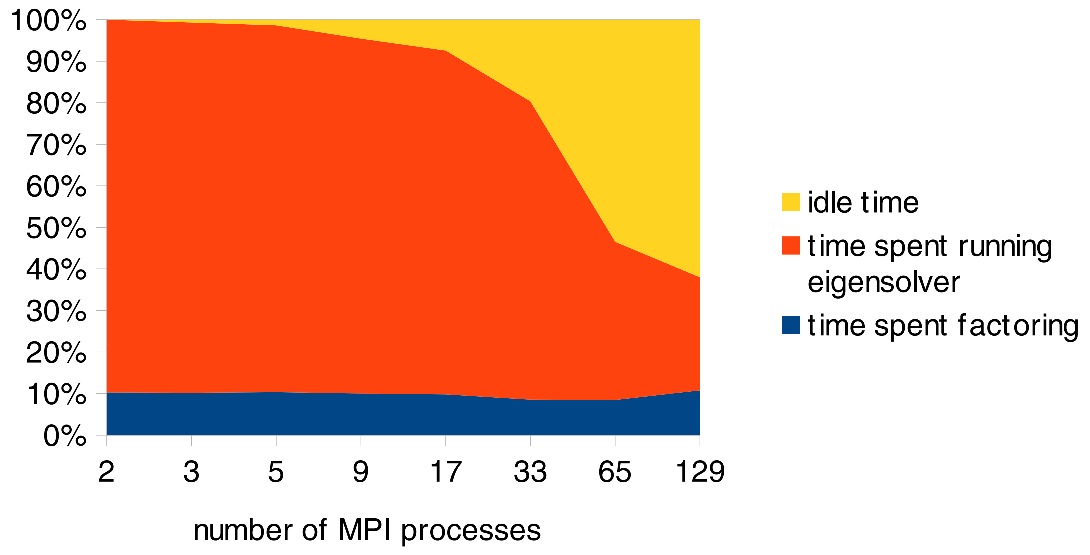


(b) Ratio of running times

Figure 8.18.: A comparison of several methods of computing a large number of eigenvalues of the Nastran benchmark (order 1.5 million)



(a) Amount of time spent in each stage (s)



(b) Percent of time spent in each stage

Figure 8.19.: Running time breakdown for TraceMin-Multisectioning on the Nastran benchmark (order 1.5 million)

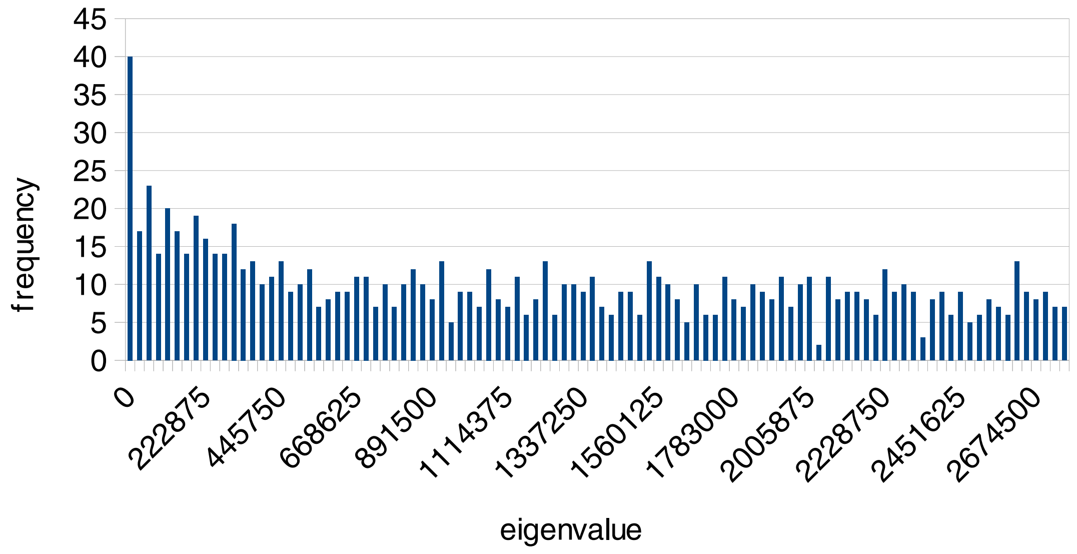


Figure 8.20.: Histogram of the eigenvalues of interest for the Nastran benchmark (order 7.2 million)

8.4.2 Nastran benchmark (order 7.2 million)

We seek all the eigenpairs in the region [-0.01,2785937.5], which contains 1000 eigenpairs (figure 8.20). FEAST failed to run on any number of nodes in this case, presumably because the complex factorization of such a large matrix took up too much memory. TraceMin performed reasonably well on this problem up to 33 nodes again (as shown in figure 8.18), then failed to continue scaling.

8.4.3 Anderson model of localization

We seek all the eigenpairs in the region [-0.01,0.01], which contains 1143 eigenpairs (figure 8.23). FEAST failed to run on a small number of nodes, again due to memory issues. It failed to scale at all from 5 to 129 nodes since the vast majority of the running time is spent on the factorization stage rather than solving linear systems; no matter how many subintervals we use, we still require the same number of

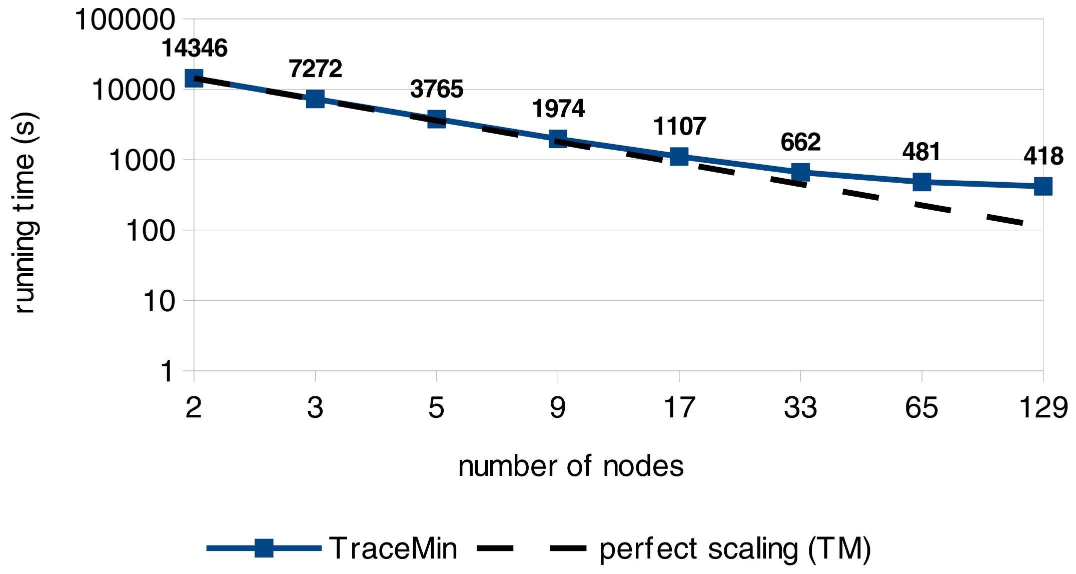
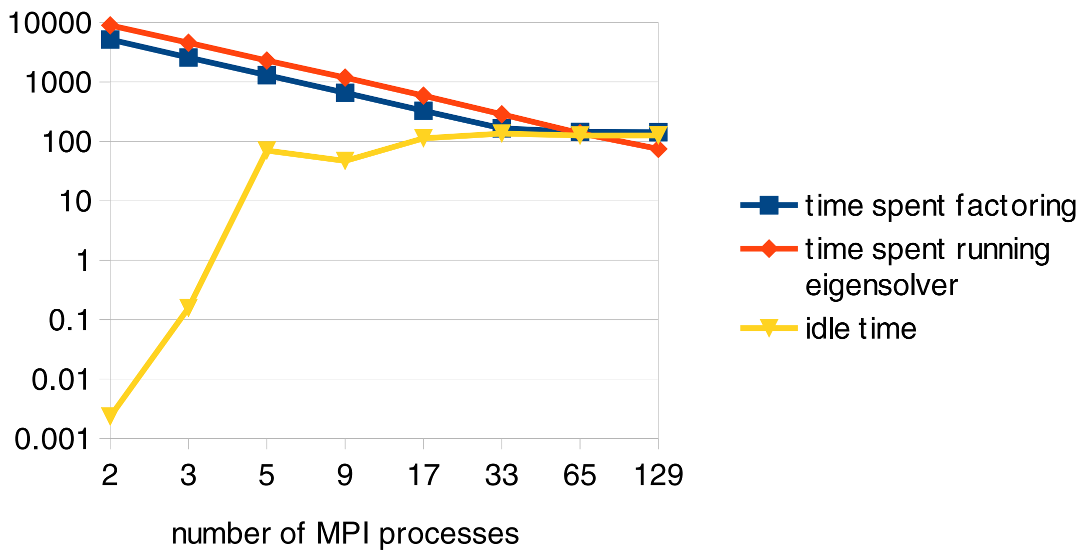
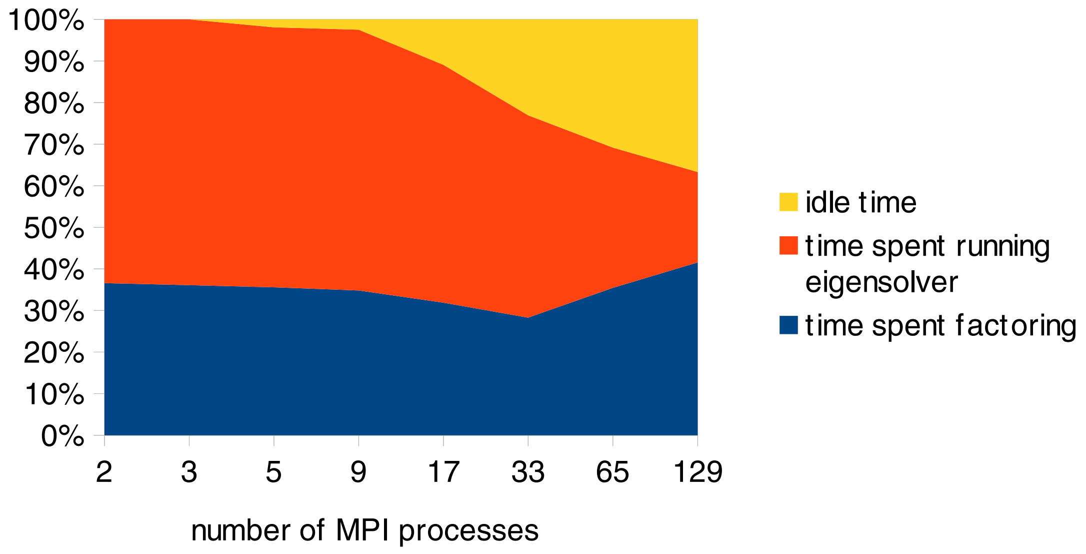


Figure 8.21.: A comparison of several methods of computing a large number of eigenvalues of the Nastran benchmark (order 7.2 million)



(a) Amount of time spent in each stage (s)



(b) Percent of time spent in each stage

Figure 8.22.: Running time breakdown for TraceMin-Multisectioning on the Nastran benchmark (order 7.2 million)

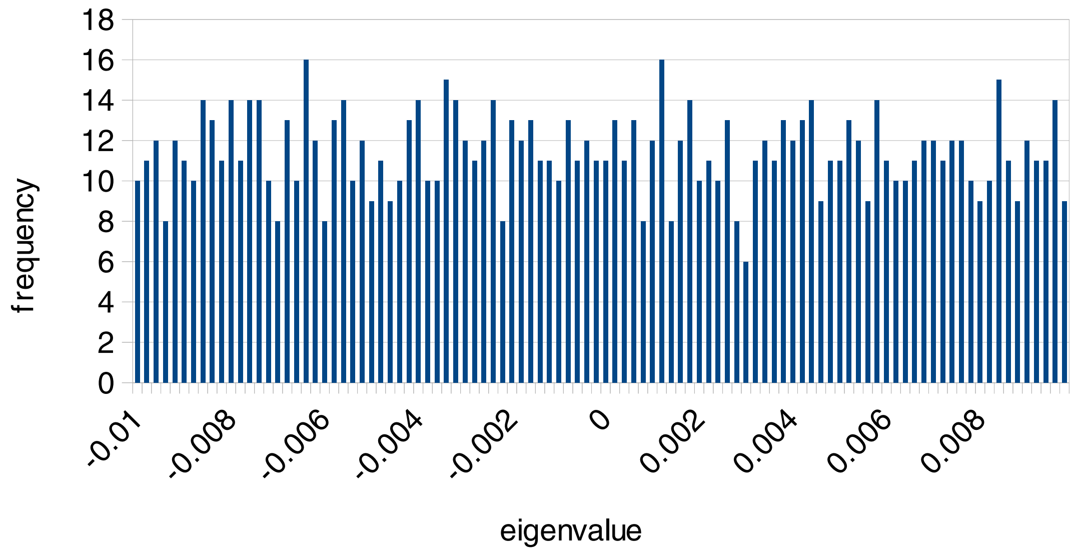


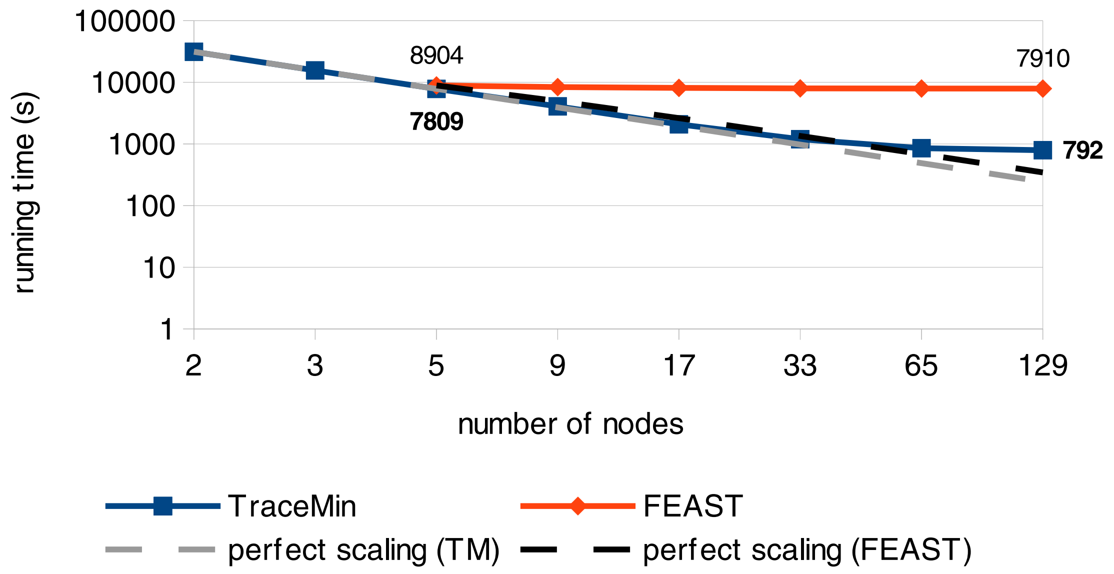
Figure 8.23.: Histogram of the eigenvalues of interest for the Anderson model

factorizations. TraceMin-Multisectioning spent most of its time in the preprocessing stage, since this matrix is so difficult to factor[4]. TraceMin-Multisectioning was still 10 times faster than FEAST on this problem.

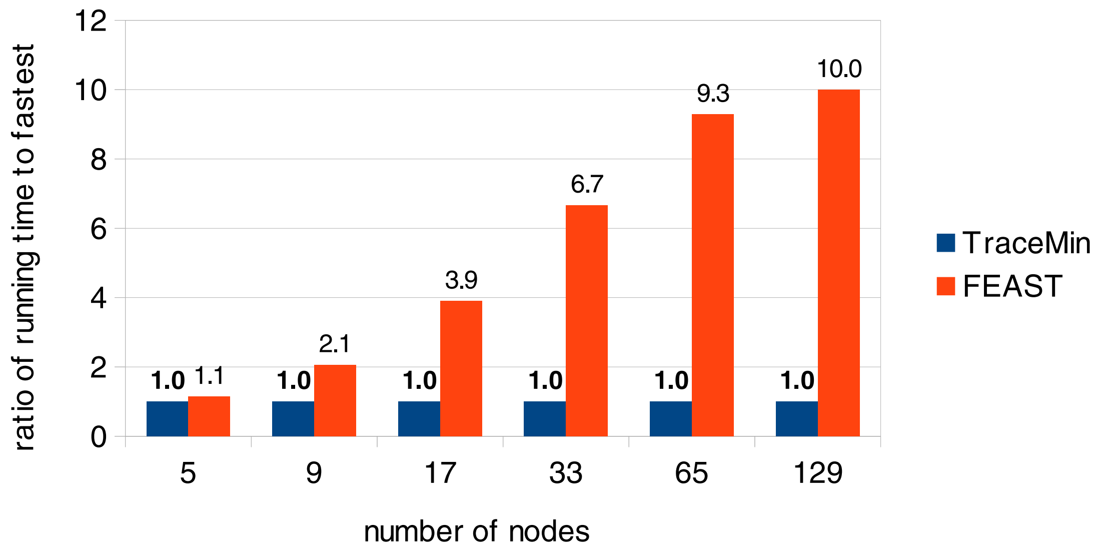
8.4.4 af_shell10

The matrix Schenk_AFE/af_shell10 (figure 8.26) arises from an AutoForm Engineering sheet metal forming simulation. We will compute all the eigenpairs in the interval [2000,2250], which contains 1045 eigenvalues (figure 8.27). In this case, TraceMin multisectioning scaled reasonably well up to 129 nodes (figure 8.28), while FEAST did not. Figure 8.29 shows that we scaled well because factorizations were cheap, and our processes did not spend a large amount of time idle.

[4]Recall that our matrix A has the same sparsity pattern as the 3D discretization of the Laplace operator with periodic boundary conditions.

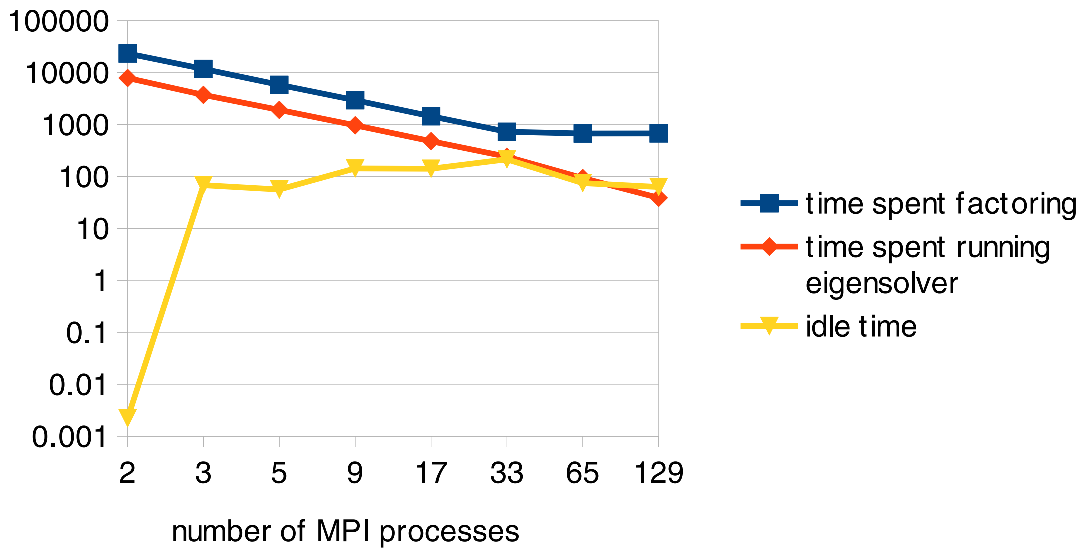


(a) Scalability comparison

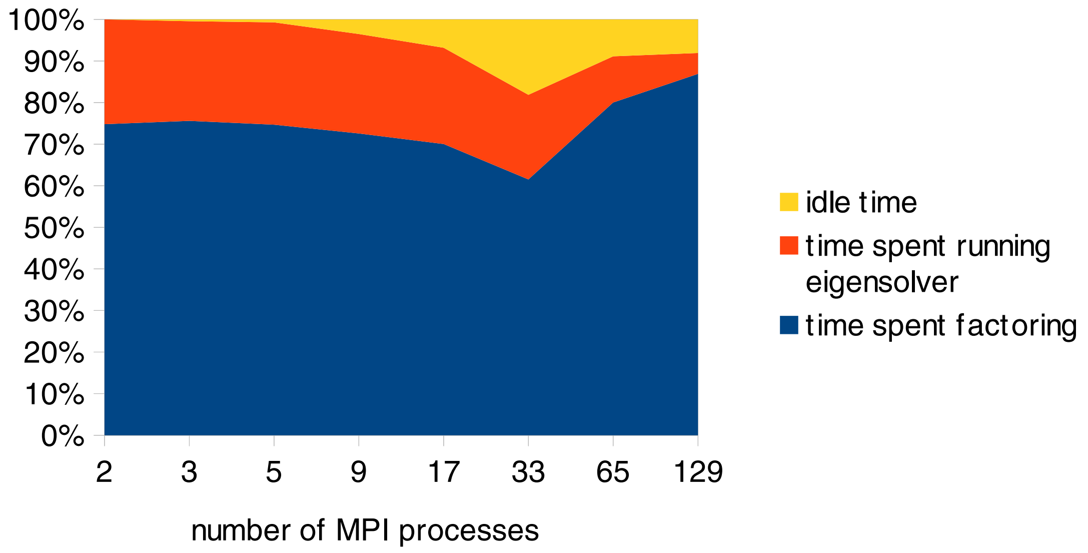


(b) Ratio of running times

Figure 8.24.: A comparison of several methods of computing a large number of eigenvalues of the Anderson model



(a) Amount of time spent in each stage (s)



(b) Percent of time spent in each stage

Figure 8.25.: Running time breakdown for TraceMin-Multisectioning on the Anderson model

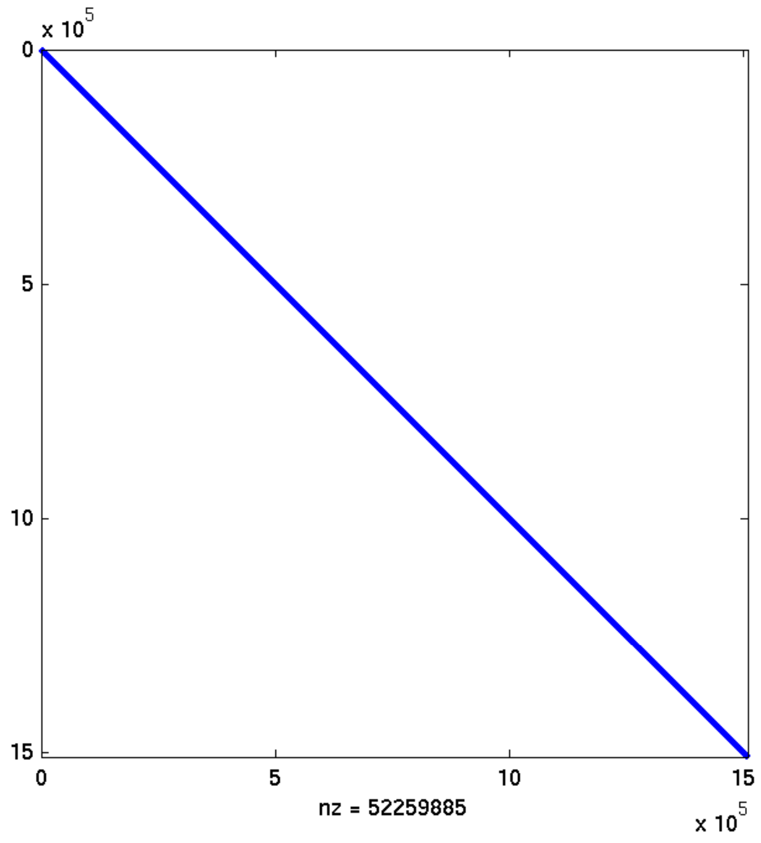


Figure 8.26.: Sparsity pattern of af_shell10

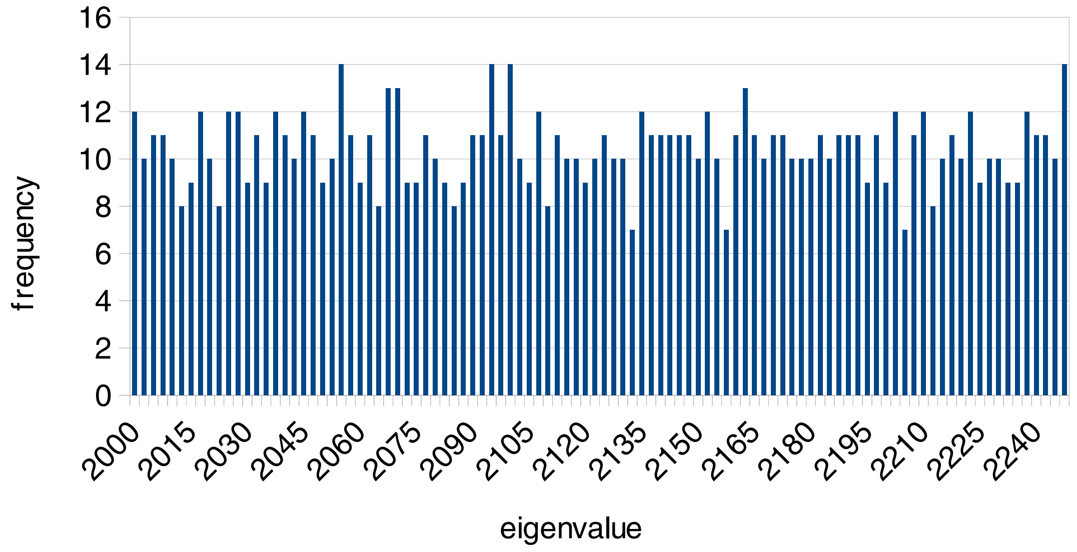
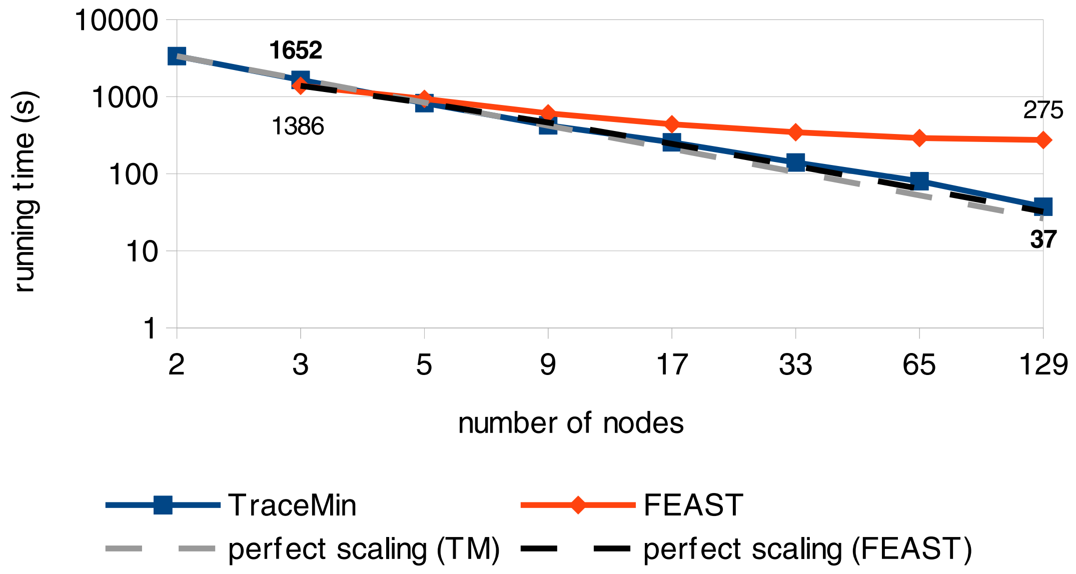
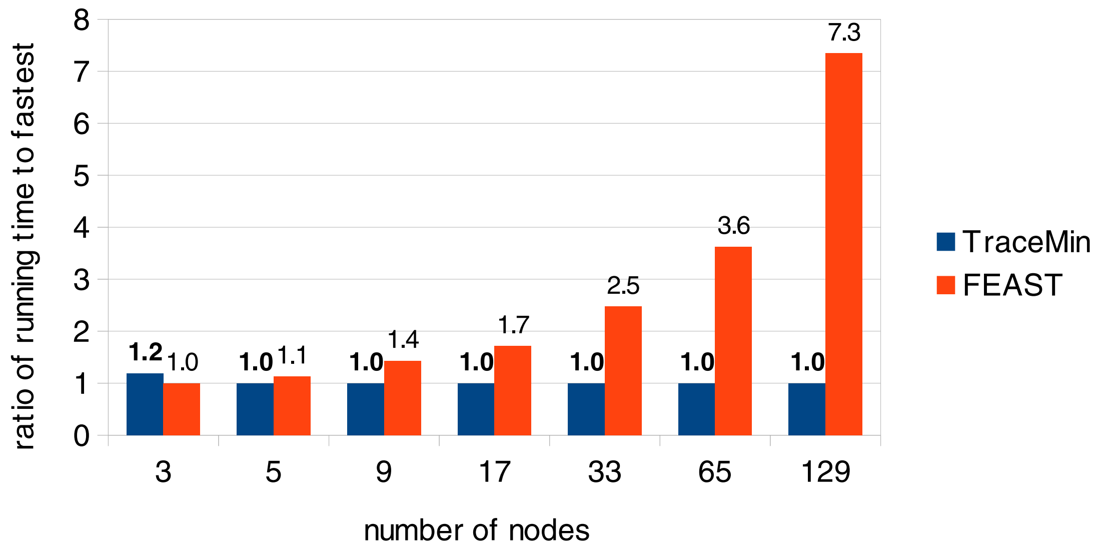


Figure 8.27.: Histogram of the eigenvalues of interest for af_shell10

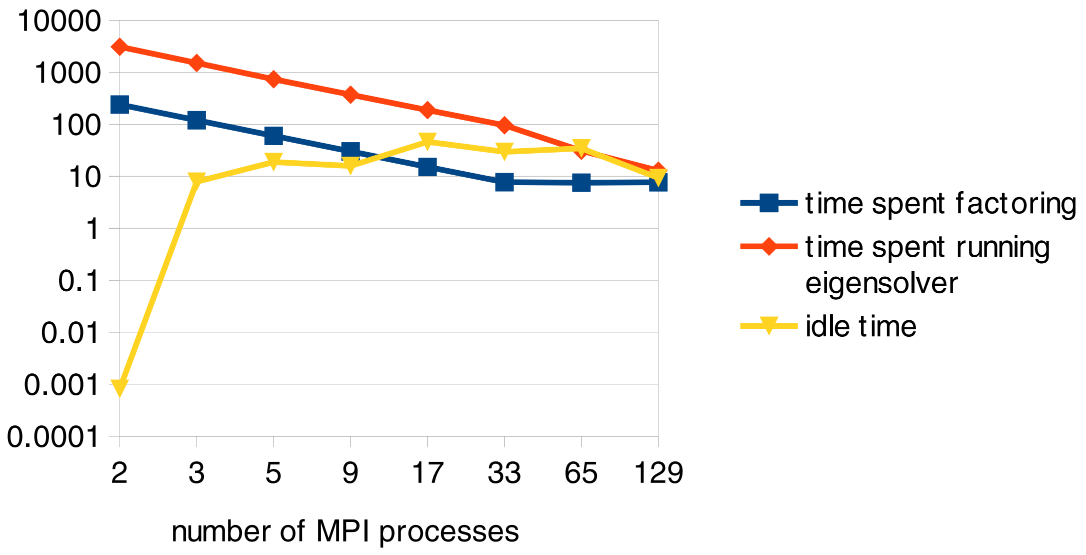


(a) Scalability comparison

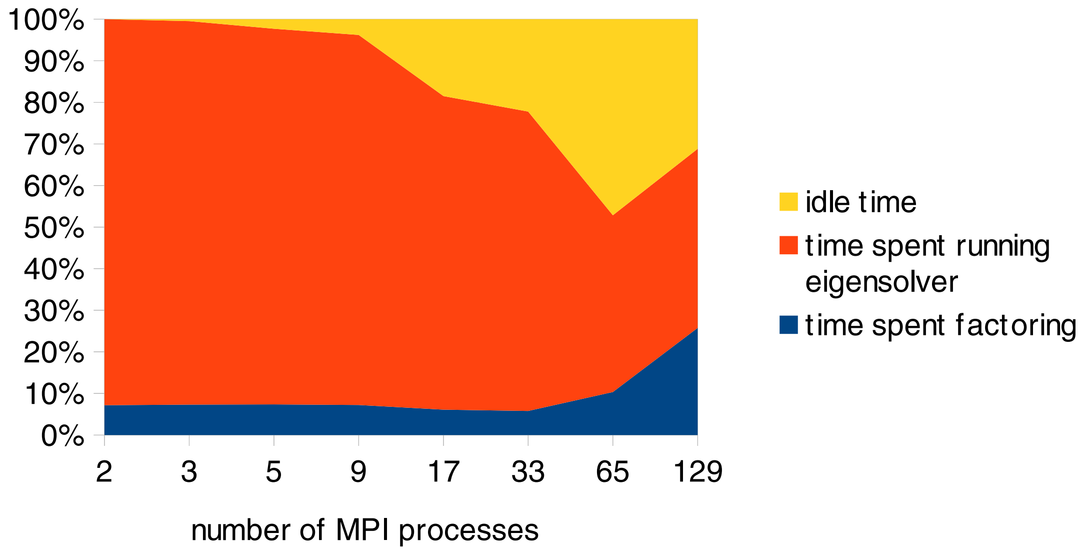


(b) Ratio of running times

Figure 8.28.: A comparison of several methods of computing a large number of eigenvalues of af_shell10



(a) Amount of time spent in each stage (s)



(b) Percent of time spent in each stage

Figure 8.29.: Running time breakdown for TraceMin-Multisectioning on af_shell10

8.4.5 dielFilterV3real

Dziekonski/dielFilterV3real is an electromagnetics matrix of order 1.1 million pictured in figure 8.30. We will compute all eigenpairs in the interval [25,50], which contains 2969 eigenpairs (figure 8.31). Note that these eigenvalues are not evenly distributed throughout the interval. FEAST fails on 2, 3, 5, 9, and 17 nodes because it ran out of space (figure 8.32); the intervals closest to 26 are very dense and require a great deal of storage. TraceMin did not run into storage issues because of its dynamic multisectioning strategy. Figure 8.33 shows that both the time spent factoring the matrix and the time spent running the eigensolver scaled as we would expect, but it did spent a large amount of time idle on large numbers of nodes, resulting in suboptimal scalabiltiy on 129 nodes. TraceMin-Multisectioning was still 3 times faster than FEAST.

8.4.6 StocF-1465

Janna/StocF-1465 is a fluid dynamics matrix of order 1.4 million pictured in figure 8.34. We will compute all eigenpairs in the interval [580,600], which contains 4150 eigenvalues (figure 8.35). Unlike the previous example, these eigenvalues are pretty evenly distributed in the interval. FEAST still failed on 2, 3, 5, and 9 nodes, since the number of vectors it needed to store was so large (figure 8.36). On 17 nodes, it has a comparable running time to TraceMin, but it failed to scale beyond that point. Once again, the factorization is the most expensive part, so the cost of running FEAST will be roughly the same regardless of how many eigenvalues an interval contains. TraceMin scaled well up to 129 nodes because in this case, there was plenty of work to go around, so the MPI processes did not spend a great deal of time idle (figure 8.37).

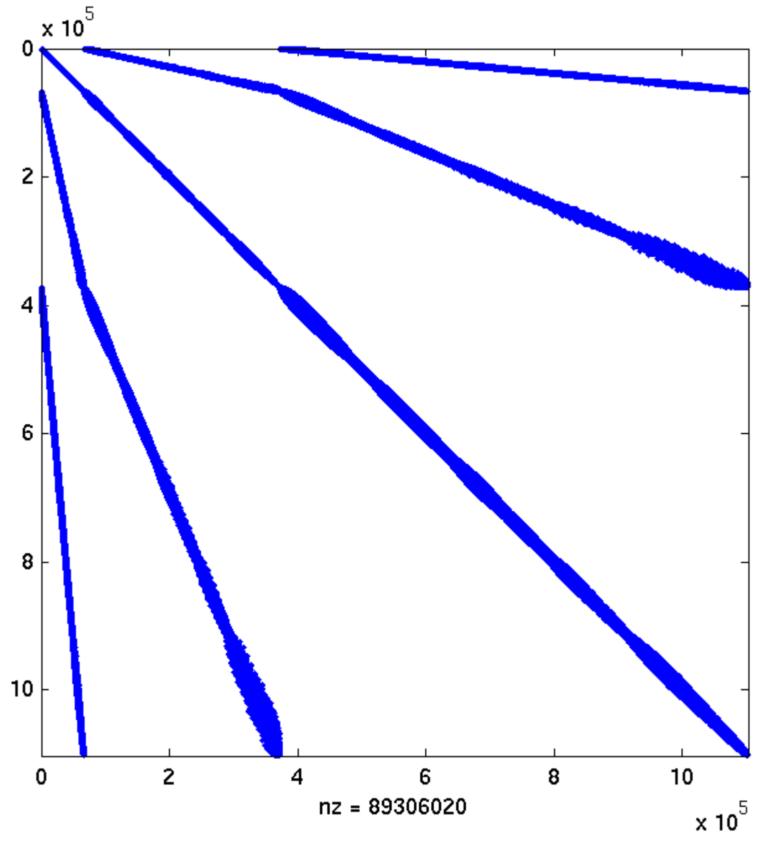


Figure 8.30.: Sparsity pattern of dielFilterV3real

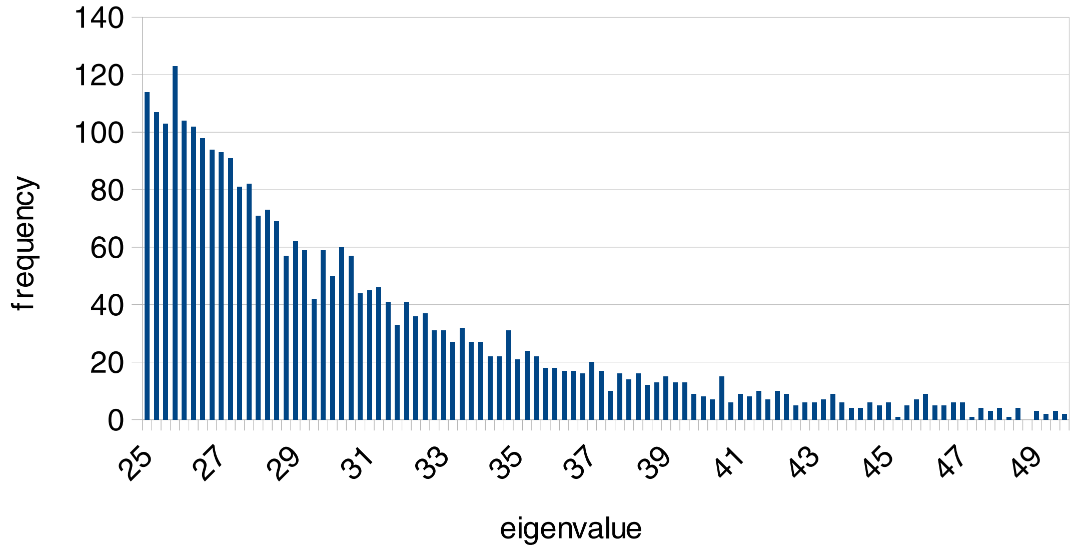
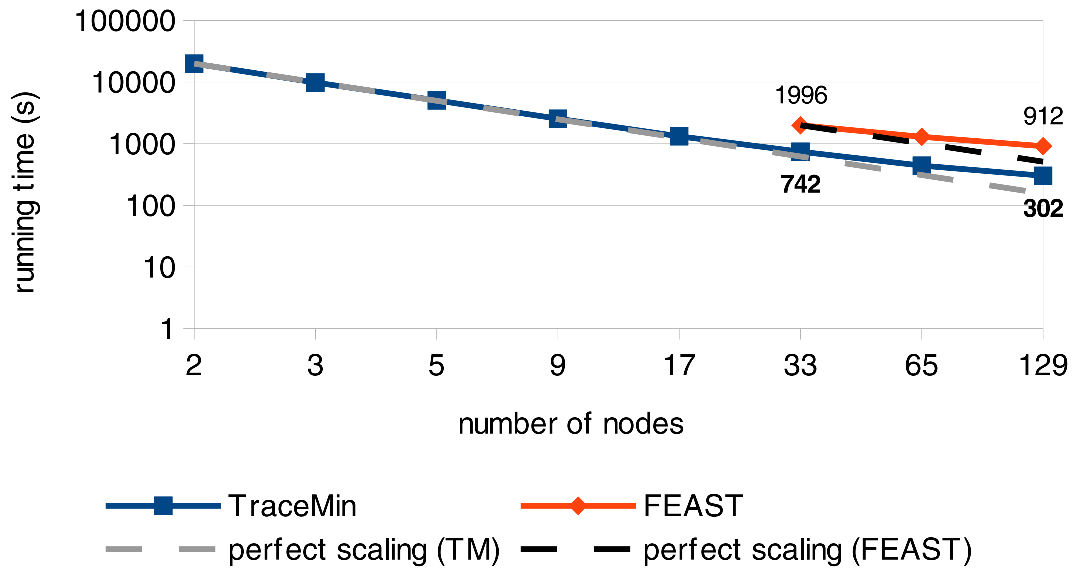
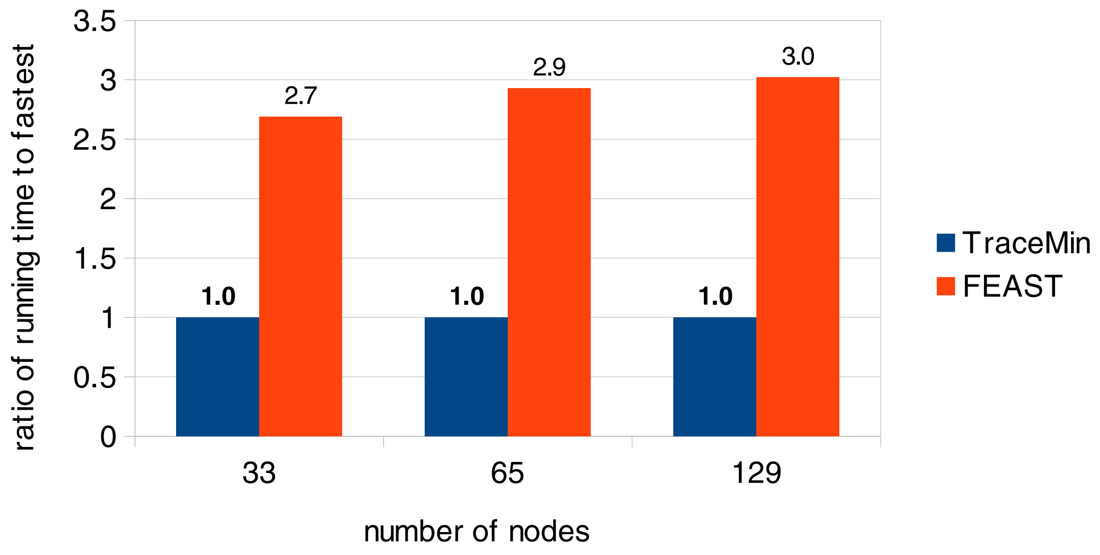


Figure 8.31.: Histogram of the eigenvalues of interest for dielFilterV3real

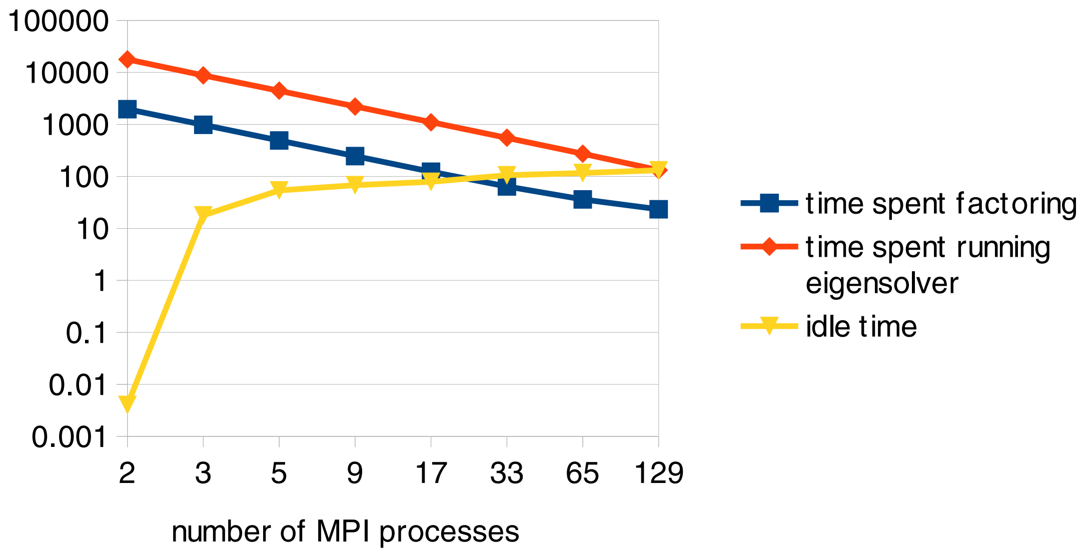


(a) Scalability comparison

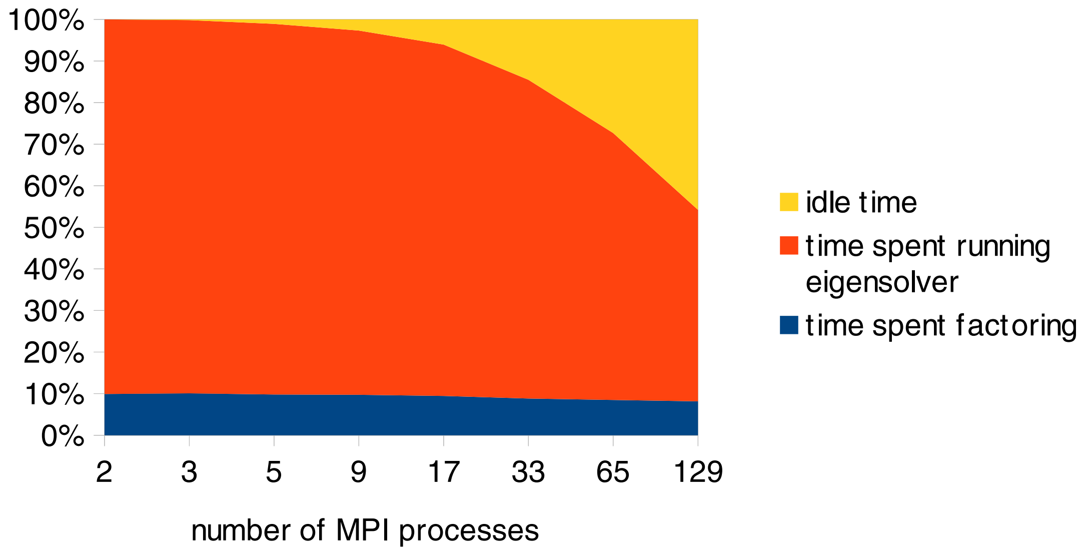


(b) Ratio of running times

Figure 8.32.: A comparison of several methods of computing a large number of eigenvalues of dielFilterV3real



(a) Amount of time spent in each stage (s)



(b) Percent of time spent in each stage

Figure 8.33.: Running time breakdown for TraceMin-Multisectioning on dielFilterV3real

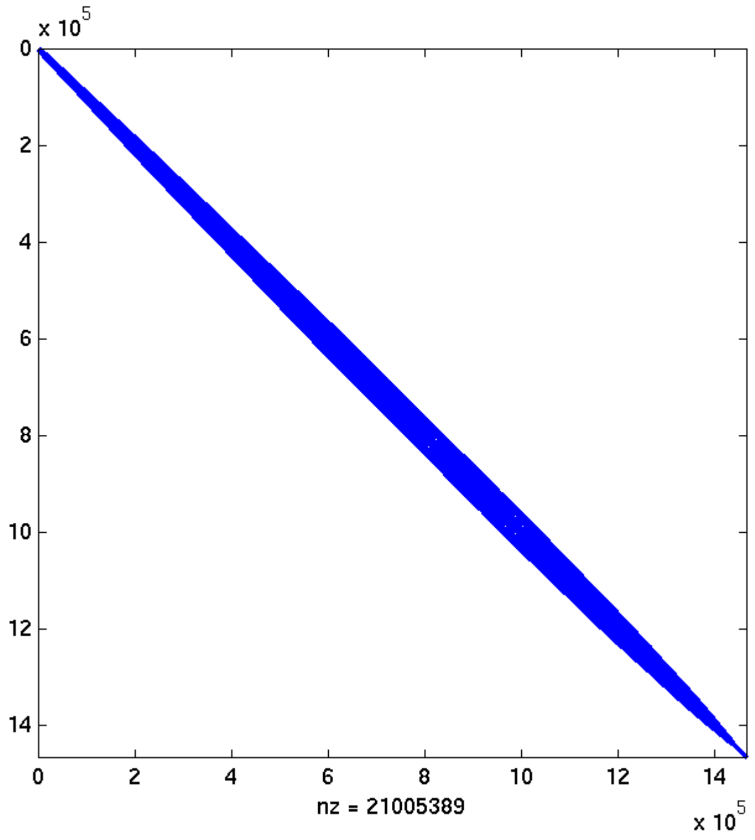


Figure 8.34.: Sparsity pattern of StocF-1465

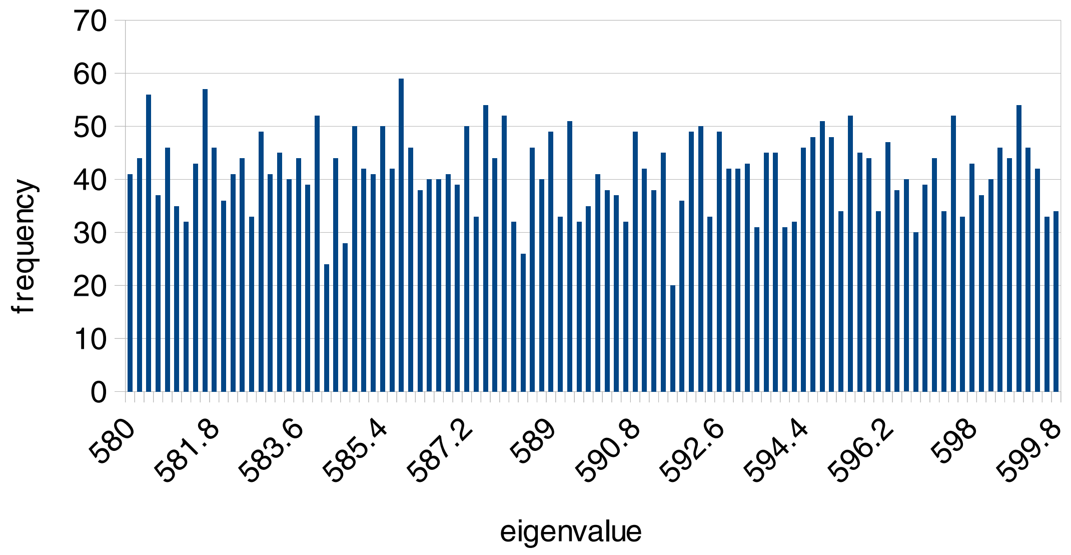
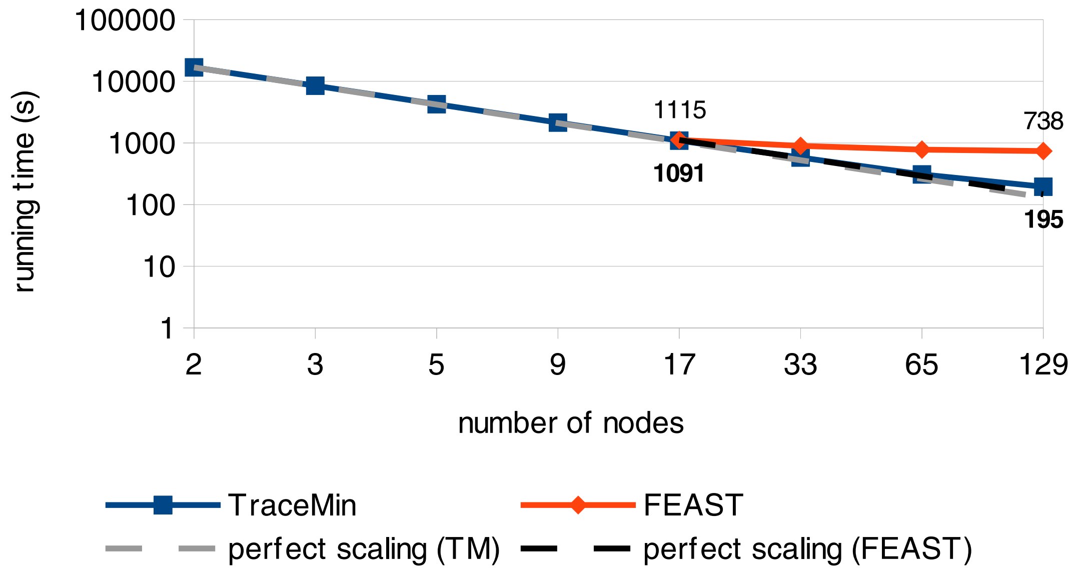
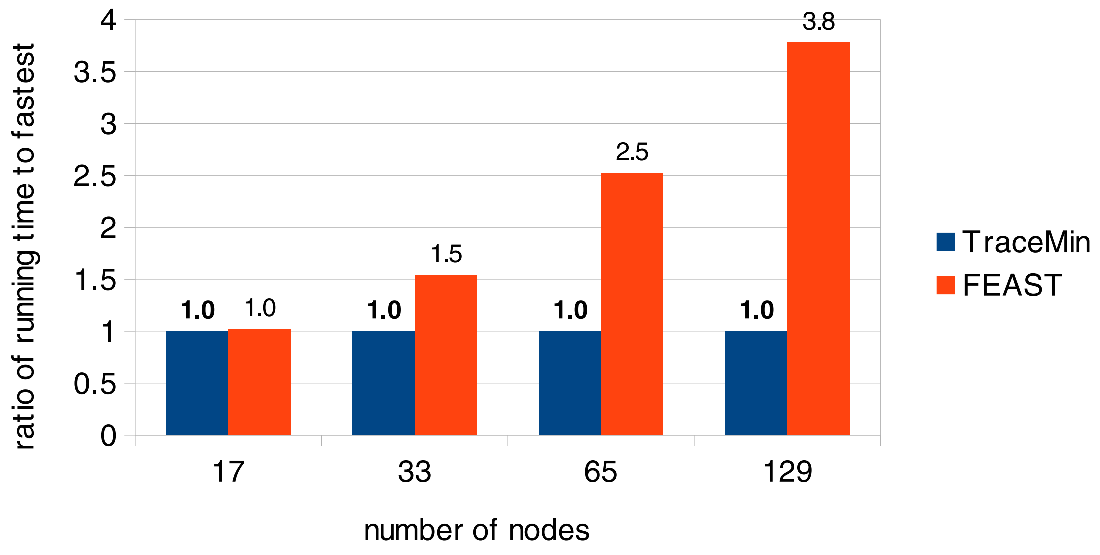


Figure 8.35.: Histogram of the eigenvalues of interest for StocF-1465

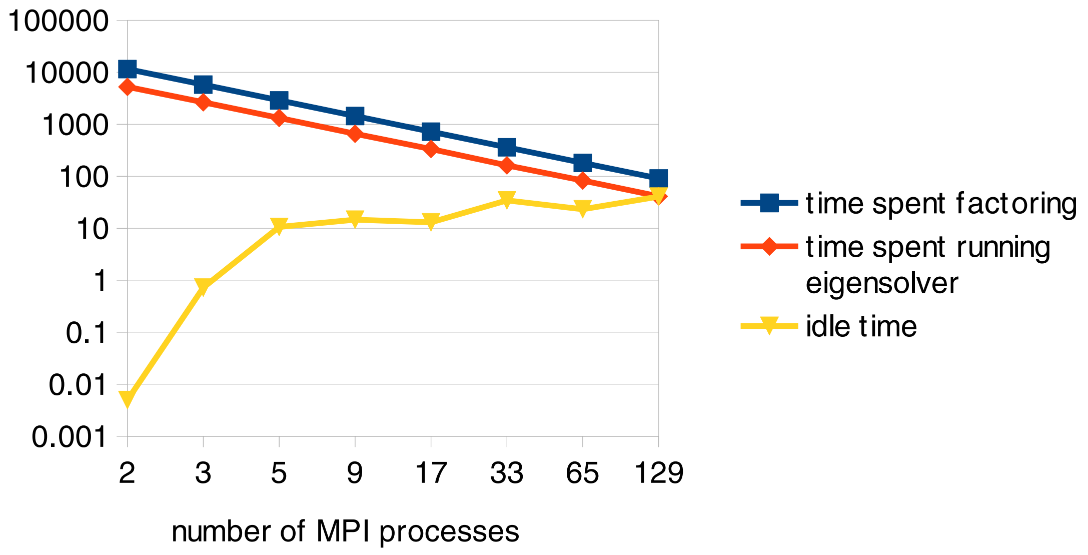


(a) Scalability comparison

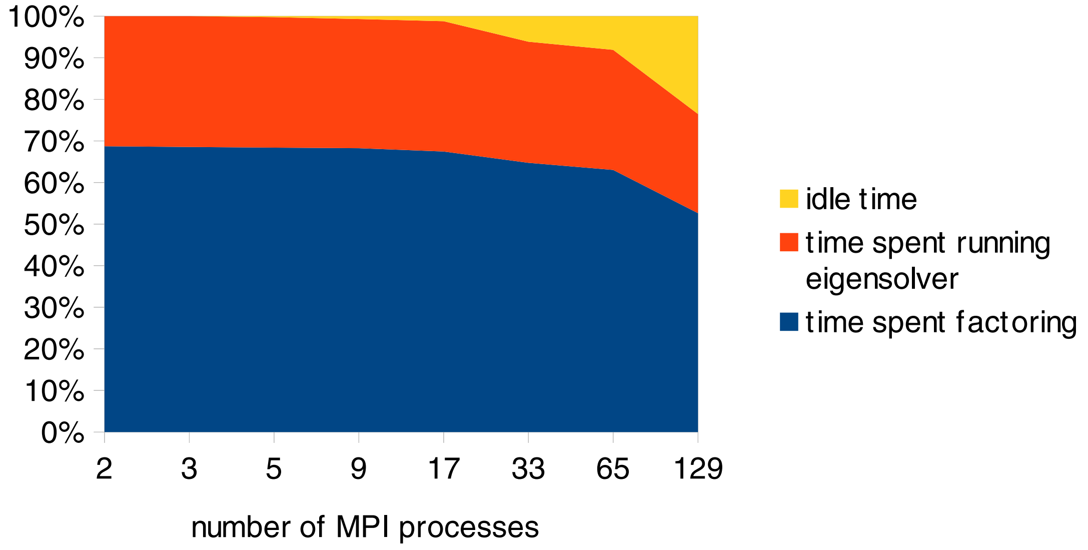


(b) Ratio of running times

Figure 8.36.: A comparison of several methods of computing a large number of eigenvalues of StocF-1465



(a) Amount of time spent in each stage (s)



(b) Percent of time spent in each stage

Figure 8.37.: Running time breakdown for TraceMin-Multisectioning on StocF-1465

9 FUTURE WORK

In this chapter, we will examine further improvements to the methods previously discussed.

9.1 Improved selection of the tolerance for Krylov solvers within TraceMin

We have seen that TraceMin is capable of converging even when we use a modest tolerance for the Krylov solver called at each iteration of TraceMin. The stricter the inner tolerance, the fewer TraceMin iterations are required; however, strict inner tolerances also increase the number of Krylov iterations required. The optimal inner tolerance is based on the ratio of the desired eigenvalue to the smallest eigenvalue outside of our desired subspace

$$\frac{\lambda_i}{\lambda_{s+1}}$$

If this ratio is large (meaning the eigenvalues are clustered), we should use a lenient inner tolerance; if the ratio is closer to 0 (meaning the eigenvalues are well separated), we should use a stricter inner tolerance. In practice, we do not know this ratio, so it is difficult to choose an effective inner tolerance. We presented a method of selecting the inner tolerance based on both the Ritz values and the current iteration number which tends to work well. However, in one of the test cases presented in the Results section, LOBPCG was faster than TraceMin-Davidson, since we solved the linear systems much more accurately than what would have been optimal. We must devote more attention to the selection of this inner tolerance to avoid such situations. Perhaps we could select the inner tolerance based on the trace reduction at each iteration of TraceMin.

9.2 Combining the strengths of TraceMin and the Riemannian Trust Region method

TraceMin uses an inexact Hessian of the generalized Rayleigh quotient (equation 9.1), which causes its linear convergence.

$$H_{TraceMin} = 2A \tag{9.1}$$

The Riemannian Trust Region method (RTR) uses an exact Hessian (equation 9.2) in order to obtain superlinear convergence.

$$H_{RTR} : S \mapsto 2\left(AS - BS\left(Y^TBY\right)^{-1} Y^TAY\right) \tag{9.2}$$

These two methods have complementary qualities. TraceMIN starts off with a sharp decrese to the trace, but the trace stagnates after a certain number of iterations (unless we use an acceleration method such as dynamic Ritz shifts). The first few steps of RTR have a difficult time exploiting a good preconditioner, since efficient preconditioned steps are likely to be rejected for falling outside the trust region [45]. Naturally, it would be beneficial to combine the strengths of these two methods. In 2004, Absil, Baker, and Gallivan found that using a few iterations of TraceMin to generate an initial subspace for RTR resulted in better convergence than either method on its own [45]. However, they only looked at one very small eigenvalue problem (that of the Calgary Olympic Saddledome arena), so this must be examined further. They also neglected to specify a heuristic method of determining when to switch from TraceMin to RTR iterations. Now that both TraceMin and RTR are implemented in Trilinos sharing a common interface, we should be able to test the combined TraceMin-RTR eigensolver and determine whether RTR is a more effective acceleration method than the dynamic Ritz shifting.

9.3 Minimizing the idle time in TraceMin-Multisectioning

Our current Fortran90-based TraceMin-Multisectioning implementation has a tiny flaw that results in extra idle time for some MPI processes, limiting its parallel scalability. MPI process 0 holds a list of work that is not currently being processed by any other node; this work (a set of subintervals) is held in a stack. When another MPI process requests a subinterval to either subdivide or run TraceMin on, process 0 will give it the top item of the stack. It will disregard any cost associated with that subinterval. As we mentioned, it is difficult to estimate the amount of work running TraceMin on an interval will require. However, it is reasonable to assume subdividing an interval containing 900 eigenvalues will be more expensive than subdividing an interval containing 50 eigenvalues. It is also important to process the interval containing 900 eigenvalues first, since it will create many new subintervals which can be treated as separate jobs. Rather than storing the additional jobs in a stack, we should be using a priority queue to ensure that large subintervals get processed first.

9.4 Removing TraceMin-Multisectioning's dependence on a direct solver

In some cases, we cannot factor the matrix $A-\sigma B$ to compute the inertia. Either there is too much fill-in, or perhaps A and B were not made explicitly available. Instead of computing the exact inertia via an expensive factorization, we can use an estimate of the eigenvalue count in an interval [48] obtained via relatively cheap matrix vector multiplications and inner products [1]. The only thing this would change about TraceMin is that it would require resilience to incorrect estimates of eigenvalue counts in the intervals. We will now discuss how we could handle these incorrect estimates.

If the estimate was too large, i.e. we expected 20 eigenvalues in the interval (a, b) and there were really only 10 in that interval, TraceMin can recover easily. Since

[1]In general, inner products are relatively expensive operations on large parallel architectures. Recall that in this multisectioning scheme, these inner products would only take place inside a single group of MPI processes; we would not be performing global reductions with *all* MPI processes.

TraceMin converges to the eigenvalues closest to our shift first, we can terminate the iterations as soon as we converge to an eigenvalue outside of the interval (a, b).

If the estimate was too small, i.e. we expected 20 eigenvalues in the interval (a, b) when there were really 100, TraceMin is still capable of recovering. If TraceMin converged to 20 eigenvalues and none of them exist outside the interval (a, b), we could assume that there were more in that interval and continue working as follows [2]. Let (c, d) be the smallest interval containing the 20 eigenvalues we found. We may now treat (a, c) and (d, b) as new intervals of interest and run a separate instance of TraceMin on each of them. To prevent TraceMin from getting confused near the edge of those intervals, we could project out the eigenvectors corresponding to the eigenvalues we found closest to the boundary of (a, b).

[2]If we got lucky and the interval legitimately contained exactly 20 eigenpairs, we could still perform this procedure. We would simply run TraceMin on the two intervals (a, c) and (d, b) until they converged to one eigenvalue outside of those intervals, then recognize that they were empty. This would create unnecessary work, but it would ensure that we did not miss any eigenvalues.

10 SUMMARY

The solution of sparse symmetric eigenvalue problems plays a significant role in many fields of computational science and engineering. Many eigensolvers require accurate solutions of linear systems in order to compute the smallest eigenpairs, which can be infeasible when the matrices are large and ill-conditioned. We presented a family of trace-minimization eigensolvers which converge even when the linear systems are solved iteratively with a modest tolerance.

First, we reviewed some applications that give rise to sparse symmetric eigenvalue problems, such as automotive engineering, condensed matter physics, and spectral reordering. Then we presented several methods of solving saddle point problems, the most important and time-consuming kernel of these trace-minimization methods. We may either use a projected-Krylov method, compute the Schur complement, or use a Krylov method with a block diagonal preconditioner.

After exploring how to solve the saddle point problems arising at each iteration of the trace-minimization eigensolvers, we described two such solvers: TraceMin and TraceMin-Davidson. TraceMin constructs its approximate eigenvectors from a subspace of constant dimension, whereas TraceMin-Davidson uses expanding subspaces. We discussed the impact of the block size and distribution of eigenvalues on the overall convergence rate, as well as how dynamic Ritz shifts can improve the rate of convergence. We also explored the impact of the tolerance used for the inner Krylov method and saw that these eigensolvers can converge even when very inexact solves are used, unlike methods such as simultaneous iteration. In addition, we studied how harmonic Ritz extraction can benefit TraceMin-Davidson in finding interior eigenvalues.

Next, we explained how to solve several types of problems with TraceMin and TraceMin-Davidson. These eigensolvers are designed to compute the smallest magnitude eigenvalues of a symmetric eigenvalue problem, but we can cause them to target alternate eigenpairs through the use of spectral transformations. We also presented two additional implementations referred to as TraceMin-Sampling and TraceMin-Multisectioning. TraceMin-Sampling is designed to compute a few eigenpairs near a large set of shifts, and TraceMin-Multisectioning was created to compute all the eigenpairs in an interval. Our multisectioning implementation uses a method similar to adaptive quadrature to subdivide the large global interval into many smaller subintervals which can be processed independently.

We then described several other popular eigensolvers such as Krylov-Schur, the Locally Optimal Block Preconditioned Conjugate Gradient method, Jacobi-Davidson, the Riemannian Trust Region method, and FEAST. After establishing how the competing eigensolvers work, we presented comparisons between those eigensolvers and our trace-minimization solvers. The results showed that TraceMin and TraceMin-Davidson are very robust, and our implementations are quite competitive with the leading software packages in terms of speed and scalabiltiy.

LIST OF REFERENCES

LIST OF REFERENCES

[1] V. Mehrmann and C. Schröder. Nonlinear eigenvalue and frequency response problems in industrial practice. *Journal of Mathematics in Industry*, 1:7, 2011.

[2] P. Anderson. Absence of diffusion in certain random lattices. *Physical Review*, 109:1492–1505, 1958.

[3] A., B. van Tiggelen, and D. Wiersma. Fifty years of Anderson localization. *Physics Today*, 62:24–29, August 2009.

[4] O. Schenk, M. Bollhoefer, and R. Roemer. On large-scale diagonalization techniques for the Anderson model of localization. *SIAM Review*, 50:91–112, 2008.

[5] U. Elsner, V. Mehrmann, F. Milde, R. Roemer, and M. Schreiber. The Anderson model of localization: a challenge for modern eigenvalue methods. *SIAM Journal on Scientific Computing*, 20:2089–2102, 1999.

[6] M. Fiedler. Algebraic connectivity of graphs. *Czechoslovak Mathematical Journal*, 23(98), 1973.

[7] E. Polizzi and A. Sameh. A parallel hybrid banded system solver: the SPIKE algorithm. *Parallel Computing*, 32(2):177–194.

[8] M. Manguoglu, A. Sameh, and O. Schenk. PSPIKE: A parallel hybrid sparse linear system solver. *Lecture Notes in Computer Science*, 5704:797–808, 2009.

[9] A. Sameh and J. Wisniewski. A trace minimization algorithm for the generalized eigenvalue problem. *SIAM Journal on Numerical Analysis*, 19(6):1243–1259, 1982.

[10] M. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of research of the National Bureau of Standards*, 49:409–436, 1952.

[11] C. Paige and M. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM Journal on Numerical Analysis*, 12:617–629, 1975.

[12] M. Murphy, G. Golub, and A. Wathen. A note on preconditioning for indefinite linear systems. *SIAM Journal on Scientific Computing*, 21:1969–1972, 1999.

[13] C. Keller, N. Gould, and A. Wathen. Constraint preconditioning for indefinite linear systems. *SIAM Journal on Matrix Analysis and Applications*, 21:1300–1317, 2000.

[14] G. Golub Z. Bai and M. Ng. Hermitian and skew-Hermitian splitting methods for non-Hermitian positive definite linear systems. *SIAM Journal on Matrix Analysis and Applications*, 24:603–626, 2003.

[15] V. Simoncini and M. Benzi. Spectral properties of the Hermitian and skew-Hermitian splitting preconditioner for saddle point problems. *SIAM Journal on Matrix Analysis and Applications*, 26:377–389, 2004.

[16] E. Beckenbach and R. Bellman. *Inequalities.* Springer, New York, 1965.

[17] A. Sameh and Z. Tong. The trace minimization method for the symmetric generalized eigenvalue problem. *Journal of Computational and Applied Mathematics*, 123(1-2):155–175, 2000.

[18] F. Bauer. Das Verfahren der Treppeniteration und verwandte Verfahren zur Lösung algebraisher Eigenwertprobleme. *Zeitschrift für angewandte Mathematik und Physik*, 8:214–235, 1957.

[19] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, editors. *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide.* SIAM, Philadelphia, 2000.

[20] M. Heroux, R. Bartlett, V. Howle, R. Hoekstra, J. Hu, T. Kolda, R. Lehoucq, K. Long, R. Pawlowski, E. Phipps, A. Salinger, H. Thornquist, R. Tuminaro, J. Willenbring, A. Williams, and K. Stanley. An overview of the Trilinos project. *ACM Transactions on Mathematical Software*, 31:397–423, 2005.

[21] S. Balay, S. Abhyankar, M. Adams, J. Brown, P. Brune, K. Buschelman, V. Eijkhout, W. Gropp, D. Kaushik, M. Knepley, L. McInnes, K. Rupp, B. Smith, and H. Zhang. PETSc web page, 2014. `http://www.mcs.anl.gov/petsc`.

[22] S. Balay, S. Abhyankar, M. Adams, J. Brown, P. Brune, K. Buschelman, V. Eijkhout, W. Gropp, D. Kaushik, M. Knepley, L. McInnes, K. Rupp, B. Smith, and H. Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.5, Argonne National Laboratory, 2014.

[23] S. Balay, W. Gropp, L. McInnes, and B. Smith. Efficienct management of parallelism in object oriented numerical software libraries. In E. Arge, A. Bruaset, and H. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhauser Press, 1997.

[24] V. Hernandez, J. Roman, and V. Vidal. SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems. *ACM Transactions on Mathematical Software*, 31(3):351–362, 2005.

[25] V. Hernandez, J. Roman, and V. Vidal. SLEPc: Scalable Library for Eigenvalue Problem Computations. *Lecture Notes in Computer Science*, 2565:377–391, 2003.

[26] J. E. Roman, C. Campos, E. Romero, and A. Tomas. SLEPc users manual. Technical Report DSIC-II/24/02 - Revision 3.5, D. Sistemes Informàtics i Computació, Universitat Politècnica de València, 2014.

[27] Anasazi examples. `trilinos.org/docs/dev/packages/anasazi/doc/html/examples.html`, March 2015.

[28] G. Golub and C. Van Loan. *Matrix Computations.* Johns Hopkins University Press, Baltimore, Maryland, 1996.

[29] R. Plemmons G. Golub and A. Sameh. *High-Speed Computing: Scientific Applications and Algorithm Design*. University of Illinois Press, Champaign, Illinois, 1988.

[30] J. Demmel, L. Grigori, M. Hoemmen, and J. Langou. Communication-optimal parallel and sequential QR and LU factorizations. Technical Report UCB/EECS-2008-89, EECS Department, University of California, Berkeley, Aug 2008.

[31] O. Schenk and K. Gärtner. Solving unsymmetric sparse systems of linear equations with PARDISO. *Future Generation Computer Systems*, 20(3):475 – 487, 2004.

[32] O. Schenk and K. Gärtner. On fast factorization pivoting methods for sparse symmetric indefinite systems. *Electronic Transactions on Numerical Analysis*, 23:158 – 179, 2006.

[33] P. Amestoy, I. Duff, J. Koster, and J.-Y. LÈxcellent. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal of Matrix Analysis and Applications*, 23(1):15–41, 2001.

[34] P. Amestoy, A. Guermouche, J.-Y. LÈxcellent, and S. Pralet. Hybrid scheduling for the parallel solution of linear systems. *Parallel Computing*, 32(2):136–156, 2006.

[35] A. Gupta and H. Avron. WSMP: Watson Sparse Matrix Package part I - direct solution of symmetric systems. Technical report, IBM T.J. Watson Research Center, 2015.

[36] C. Baker, U. Hetmaniuk, R. Lehoucq, and H. Thornquist. Anasazi software for the numerical solution of large-scale eigenvalue problems. *ACM Transactions on Mathematical Software*, 36(3):13:1–13:23, July 2009.

[37] E. Polizzi. Density-matrix-based algorithms for solving eigenvalue problems. *Physical Review B.*, 79, 2009.

[38] E. Polizzi. A high-performance numerical library for solving eigenvalue problems: FEAST solver v2.0 user's guide. *Computing Research Repository*, abs/1203.4031, 2012.

[39] G. Stewart. A Krylov-Schur algorithm for large eigenproblems. *SIAM Journal on Matrix Analysis and Applications*, 23:601–614, 2000.

[40] Y. Zhou and Y. Saad. Block Krylov-Schur method for large symmetric eigenvalue problems. *Numerical Algorithms*, 47:341–359, 2008.

[41] A. Knyazev. Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method. *SIAM Journal on Scientific Computing*, 23(2):517–541, 2001.

[42] I. Lashuk A. Knyazev, M. Argentati and E. Ovtchinnikov. Block Locally Optimal Preconditioned Eigenvalue Xolvers (BLOPEX) in Hypre and PETSc. *SIAM Journal on Scientific Computing*, 29(5):2224–2239, 2007.

[43] G. Sleijpen and H. van der Vorst. A Jacobi–Davidson iteration method for linear eigenvalue problems. *SIAM Journal on Matrix Analysis and Applications*, 17:401–425, 1996.

[44] P. Absil, C. Baker, and K. Gallivan. Trust-region methods on Riemannian manifolds. 2004.

[45] P. Absil, C. Baker, K. Gallivan, and A. Sameh. Adaptive model trust region methods for generalized eigenvalue problems. Technical report, Florida State University, 2004.

[46] A. Baggag and A. Sameh. A nested iterative scheme for indefinite linear systems in particulate flows. *Computer Methods in Applied Mechanics and Engineering*, 193:1923–1957, 2004.

[47] T. Davis and Y. Hu. The university of florida sparse matrix collection. *ACM Transactions on Mathematical Software*, 38:1–25, 2011.

[48] E. Di Napoli, E. Polizzi, and Y. Saad. Efficient estimation of eigenvalue counts in an interval. preprint, `http://arxiv.org/abs/1308.427`.

VITA

VITA

Alicia Marie Klinvex was born in Pittsburgh, Pennsylvania in 1986. In May 2008, she received her bachelor's degree in computer science from The Pennsylvania State University's Erie campus. She then went on to earn a PhD from Purdue University's Department of Computer Science in May 2015. After graduation, she worked as a postdoctoral researcher at Sandia National Laboratories as a developer for the Trilinos project.