Parallel Solution of the Symmetric Tridiagonal Eigenproblem

Elizabeth R. Jessup

Research Report YALEU/DCS/RR-728
October 1989

The author was supported in part by an IBM Fellowship and in part by the Office of Naval Research under contracts N000014-82-K-0184 and N00014-85-K-0461.

ABSTRACT

Parallel Solution of the Symmetric Tridiagonal Eigenproblem

Elizabeth Redding Jessup

Yale University

1989

This thesis discusses methods for computing all eigenvalues and eigenvectors of a symmetric tridiagonal matrix on a distributed-memory MIMD multiprocessor. Only those techniques having the potential for both high numerical accuracy and significant large-grained parallelism are investigated. These include the QL method or Cuppen's divide and conquer method based on rank-one updating to compute both eigenvalues and eigenvectors, bisection to determine eigenvalues, and inverse iteration to compute eigenvectors.

To begin, the methods are compared with respect to computation time, communication time, parallel speedup, and accuracy. Experiments on an iPSC hypercube multiprocessor reveal that Cuppen's method is the most accurate approach, but bisection with inverse iteration is the fastest and most parallel. Because the accuracy of the latter combination is determined by the quality of the computed eigenvectors, the factors influencing the accuracy of inverse iteration are examined. This includes, in part, statistical analysis of the effects of a starting vector with random components. These results are used to develop an implementation of inverse iteration producing eigenvectors with lower residual error and better orthogonality than those generated by the EISPACK routine TINVIT. This thesis concludes with adaptations of methods for the symmetric tridiagonal eigenproblem to the related problem of computing the singular value decomposition (SVD) of a bidiagonal matrix.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis discusses efficient serial and parallel methods for computing all eigenvalues and eigenvectors of a real symmetric tridiagonal matrix $T = U\Lambda U^T$ to high numerical accuracy. These methods are also applied to the related problem of computing the singular value decomposition (SVD) of a real bidiagonal matrix $B = Y\Sigma X^T$.

The methods examined exhibit large-grained parallelism on the order of vector-vector, matrix-vector, or small-scale matrix-matrix operations suitable for implementation on a distributed-memory MIMD (*multiple instruction, multiple data*) multiprocessor with scalar processors. The algorithms employ static processor scheduling.

1.1 Background

Singular value decompositions and symmetric eigenproblems occur in a wide variety of applications. In many cases, the problems are of very large order. For example, properties of certain quantum dynamical systems can be determined through statistical analysis of quantities computed from the eigenvalues or eigenvectors of symmetric matrices associated with those systems [44, 45]. In order to discern structure in the distributions of these quantities, it is often necessary to determine to high accuracy the full eigensystems of matrices of order 1000 or more [43]. Because these computations constitute a time-consuming process demanding extensive memory, they illustrate the need for parallel eigensolvers. Large order problems in real-time signal processing [69], among other applications, motivate parallel methods for the SVD.

Matrices arising in such applications are sometimes tridiagonal [53] and often banded [44]. In addition, tridiagonal and bidiagonal matrices arise in the solution of more general problems. That is, full eigendecompositions of dense matrices are usually computed by Jacobi methods [33] or by direct reduction of A to symmetric tridiagonal form T by Givens rotations or Householder reflections followed by computation of the eigendecomposition of T [78]. When the latter is implemented in exact arithmetic, the eigenvalues of A equal the eigenvalues of T, and the eigenvectors of A can be determined directly from those of T. Sparse eigenproblems are often handled by the Lanczos method [59] which itself produces symmetric tridiagonal eigenproblems. The SVD of a matrix can be computed by the Jacobi method [33] or by reducing the matrix to bidiagonal form and solving the bidiagonal problem [78]. The remainder of this thesis concerns only methods for the reduced bandwidth problems.

Methods for the symmetric tridiagonal eigenproblem include the QL and QR methods [9], divide and conquer strategies [13, 52], Toda flow [16, 74], Rayleigh quotient iteration [76], and bisection with inverse iteration [4, 8, 55, 59, 76]. Multisection [5, 55] may be used in place of bisection and subspace iteration [7, 59] in place of inverse iteration. Methods for the bidiagonal singular value problem include the Golub-Reinsch QR algorithm [34], divide and conquer techniques [2, 46], and methods that convert the $n \times n$ singular value problem to a $2n \times 2n$ symmetric tridiagonal eigenproblem.

All of these methods have been implemented in parallel. Implementations of Givens' and Householder's methods for bandwidth reduction are presented for systolic arrays in [37, 39] and for shared-memory multiprocessors and processor arrays in [23, 49, 50, 66]. Toda flow techniques, Cuppen's divide and conquer method, bisection, multisection, and inverse iteration have been implemented for shared-memory multiprocessors and their simulators [8, 18, 24, 55, 74], for the ICL DAP [4, 5], and for the Illiac IV [38]. A parallel QR algorithm for the symmetric tridiagonal eigenproblem is presented in [65]. Hybrid parallel methods involving Rayleigh quotient iteration and inverse iteration [67, 73] have been implemented for shared-memory multiprocessors [57, 56].

Methods for the symmetric tridiagonal eigenproblem to be considered in this thesis are the QL method with Wilkinson's shift or the perfect shift, Cuppen's divide and conquer technique, and bisection with inverse iteration. The methods for the bidiagonal singular value problem are the Golub-Reinsch QR method [35], the new divide and conquer technique from [46], and bisection with inverse iteration [76].

The remaining methods are not accurate or are variations of the ones studied. Toda flow methods give only approximate solutions [74]. The divide and conquer techniques presented in [2, 52] are not studied specifically, but they possess the same recursive structure as divide and conquer techniques to be examined. Hybrid Rayleigh quotient iteration methods, multisection, and subspace iteration are also not treated although many of the results about bisection and multisection apply, and the hybrid and generalized methods might be used in place of bisection and inverse iteration.

1.2 Outline of the Thesis

This thesis shows that bisection with inverse iteration is generally the fastest and most parallel accurate approach to the symmetric tridiagonal eigenproblem and the bidiagonal singular value problem on a statically scheduled, distributed-memory multiprocessor. In addition, bisection with inverse iteration competes with the divide and conquer techniques for the fastest accurate serial method for both problems. The dissertation proceeds as follows:

Chapter 2 outlines assumptions, notation, and criteria for evaluating the numerical methods. Chapter 3 describes the hypercube multiprocessor used in the experiments.

Chapter 4 reviews the QL method, the divide and conquer method, and bisection with inverse iteration for the symmetric tridiagonal eigenproblem. Chapter 5 compares their accuracies and parallel efficiencies on the iPSC/1-d5M hypercube multiprocessor. The experiments show that the parallel implementation of inverse iteration, which is based on the EISPACK routine TINVIT, does not produce highly accurate eigenvectors.

Chapter 6 identifies the factors influencing accuracy of inverse iteration and develops a more accurate implementation. Chapter 7 presents a statistical justification of some features of the new implementation.

Chapter 8 addresses the computation of the SVD of a bidiagonal matrix.

Portions of this thesis have been previously published in [40, 41, 46, 47].

Chapter 2

Preliminaries

This thesis compares methods for the symmetric tridiagonal eigenproblem and the bidiagonal singular value problem in terms of runtime, parallel efficiency, and accuracy. This chapter describes the test problems and measures of quality employed in the experiments and their analysis.

Notation is introduced in Section 2.1 along with assumptions about the matrices used. Definitions of and theoretical results about the measures of quality employed are presented in Section 2.2. The symmetric tridiagonal matrices actually used to test the eigensolvers and the bidiagonal ones used to test the methods for computing the singular value decomposition are given in Section 2.3.

2.1 Notation and Assumptions

Unless otherwise specified, matrices are represented by upper case Roman letters, column vectors by lower case Roman letters, and scalars by lower case Greek letters. A superscript T denotes transpose. A circumflex denotes a computed quantity. All quantities are assumed to be real.

T denotes an $n \times n$ symmetric tridiagonal matrix having the eigendecomposition $T = U \Lambda U^T$, where Λ is an $n \times n$ diagonal matrix with the eigenvalues $\lambda_1 \geq \ldots \geq \lambda_n$ as its diagonal elements. The $n \times n$ matrix U is orthogonal and has as its columns the eigenvectors $u_1, \ldots, u_n$. The matrix $T = [\beta, \alpha, \beta]$ has all diagonal elements equal to α and all off-diagonal elements equal to β. The matrix

$T = [\beta, u, \beta]$ has the vector u on its diagonal and all off-diagonal elements equal to β.

The eigenvalue of largest magnitude $\max(|\lambda_1|, |\lambda_n|)$ is written $|\lambda|_{max}$. The computed value $|\hat{\lambda}|_{max}$ approximates $\| T \|_2$.

B denotes an $n \times n$ upper bidiagonal matrix having the singular value decomposition $B = Y\Sigma X^T$. The matrix Σ is diagonal with the singular values $\sigma_1 \geq \ldots \geq \sigma_n \geq 0$ as its diagonal elements. The columns of $Y = (y_1, \ldots, y_n)$ are the left singular vectors of B; the columns of $X = (x_1, \ldots, x_n)$ are the right singular vectors of B. Y and X are both $n \times n$ orthogonal matrices. The upper bidiagonal matrix $B = [\alpha, \beta]$ has all diagonal elements equal to α and all elements on the first superdiagonal equal to β. The bidiagonal matrix $B = [\beta, u]$ has the vector u on its first super-diagonal and all diagonal elements equal to β.

The computed quantity $\hat{\sigma}_1$ approximates $\| B \|_2$.

The methods derived for a square bidiagonal matrix can be applied to an $m \times n$ bidiagonal matrix $\bar{B}$. When $m \geq n$, $\bar{B}$ is factored into

$$\bar{B} = \begin{pmatrix} I \\ 0 \end{pmatrix} B$$

with $m \times n$ $\begin{pmatrix} I \\ 0 \end{pmatrix}$. If $B = Y\Sigma X^T$, then $\bar{B} = \bar{Y}\bar{\Sigma}\bar{X}^T$ with $\bar{\Sigma} = \Sigma$, $\bar{X} = X$, and $\bar{Y} = \begin{pmatrix} I \\ 0 \end{pmatrix} Y$. When $m \leq n$,

$$\bar{B} = (\, I \quad 0\,)\, B = (\, I \quad 0\,)\, Y\Sigma X^T$$

with $m \times n$ $(\, I \quad 0\,)$, $(\, I \quad 0\,)\,\bar{\Sigma} = \Sigma$, $\bar{Y} = (\, I \quad 0\,)\, Y \begin{pmatrix} I \\ 0 \end{pmatrix}$, and $\bar{X} = X$.

It is assumed that each tridiagonal matrix T or bidiagonal matrix B is unreduced, meaning that none of the immediate sub- or superdiagonal elements of T is zero and none of the immediate superdiagonal elements of B is zero. If not, the matrix would consist of a direct product of disjoint, lower order matrices whose eigendecompositions or SVD's could be computed independently. Note that while an unreduced tridiagonal matrix has distinct eigenvalues in exact arithmetic, it may still have computationally coincident ones in finite precision.

2.2 Measures of the Quality of a Method

In this thesis, methods are compared experimentally in terms of runtime (including communication, if any), parallel efficiency, and accuracy.

The parallel speedup is determined in two ways. First, the time T_1 to solve the problem by a sequential implementation of the method on one processor is divided by the time T_p to solve the same problem by the same method on p processors. This speedup, denoted $\mathcal{S} = \frac{T_1}{T_p}$, quantifies the degree of parallelism inherent in a method. A speedup near the number of processors ($\mathcal{S} \approx p$) indicates that all processors are kept largely busy during the parallel solution of the problem and that little additional overhead (communication or redundant computation) is incurred.

The second speedup measure is $s = \frac{t_1}{T_p}$ where t_1 is the time to solve the problem by the *fastest* serial method. The speedup s reveals the maximum gain in speed possible by using the parallel method in place of a serial one on a single processor. Note that $\mathcal{S}$ can never be larger than s.

The accuracy of a method is determined by the residual error in the computed solutions and by the orthogonality of the computed eigenvectors or singular vectors. For the symmetric tridiagonal matrix T with computed eigendecomposition $\hat{U}\hat{\Lambda}\hat{U}^T$, the errors are given by

$$\begin{aligned} \mathcal{R} &= \frac{1}{|\hat{\lambda}|_{max}} \max_i \| T\hat{u}_i - \hat{\lambda}_i\hat{u}_i \|_2 \\ \mathcal{O} &= \| \hat{U}^T\hat{U} - I \|_\infty, \end{aligned}$$

The residual error is thus determined by the largest residual error for any single computed eigenpair. The norm used to measure the orthogonality is fast to compute.

For the bidiagonal singular value problem $B = \hat{Y}\hat{\Sigma}\hat{X}^T$, the errors are given by

$$\mathcal{R}_X = \frac{1}{\hat{\sigma}_1} \max_i \| B\hat{x}_i - \hat{\sigma}_i\hat{y}_i \|_2$$

$$\begin{aligned}\mathcal{R}_Y &= \frac{1}{\hat{\sigma}_1}\max_i \| \hat{y}_i^T B - \hat{\sigma}_i \hat{x}_i^T \|_2 \\ \mathcal{O}_X &= \| \hat{X}^T \hat{X} - I \|_\infty \\ \mathcal{O}_Y &= \| \hat{Y}^T \hat{Y} - I \|_\infty\end{aligned}$$

Theorems 2.2.1 and 2.2.2 below show that if residuals and orthogonality measures are small, then the computed eigendecomposition or singular value decomposition has high absolute accuracy. The proofs of these theorems depend on Lemmas 2.2.1 and 2.2.2 which establish necessary properties of the matrix norms.

Lemma 2.2.1 *[33]*

$$\| A \|_2 \leq \sqrt{n} \max_i \| Ae_i \|_2,$$

$$\frac{1}{\sqrt{n}} \| A \|_\infty \leq \| A \|_2 \leq \sqrt{n} \; \| A \|_\infty.$$

Lemma 2.2.2 *(See Theorem 2.10 in Chapter 4 of [71].)*

$$\| AA^T \|_2 = \| A \|_2{}^2.$$

Lemma 2.2.3 *Suppose V is a square matrix such that $E = V^T V - I$ with $\| E \|_2 \leq \bar{\epsilon}$, then $\bar{E} = VV^T - I$ with $\| \bar{E} \|_2 \leq \bar{\epsilon}$.*

Proof: Let $V = Y \Sigma X^T$ be the singular value decomposition of V, then

$$V^T V - I = X \Sigma^2 X^T - I = X \Sigma^2 X^T - XX^T = X(\Sigma^2 - I)X^T.$$

Similarly,

$$VV^T - I = Y(\Sigma^2 - I)Y^T.$$

Because X and Y are orthogonal matrices,

$$\| E \|_2 = \| V^T V - I \|_2 = \| VV^T - I \|_2 = \| \bar{E} \|_2.$$

■

Theorem 2.2.1 below shows that if the residual $\mathcal{R}$ and orthogonality $\mathcal{O}$ are small, then $\hat{U} \hat{\Lambda} \hat{U}^T$ is the exact eigendecomposition of a matrix $T + E$ nearly equal to T. E is neither symmetric nor tridiagonal in general.

Theorem 2.2.1 *Let $\hat{U}\hat{\Lambda}\hat{U}^T$ be the computed eigendecomposition of a symmetric tridiagonal matrix T. If $\mathcal{R} \le \epsilon_1$, and $\mathcal{O} \le \epsilon_2$, then there exists a matrix E such that*

$$T + E = \hat{U}\hat{\Lambda}\hat{U}^T, \ \textit{and} \ \ \| E \|_2 \le \sqrt{n}[|\lambda|_{max}\epsilon_2 + |\hat{\lambda}|_{max}\epsilon_1\sqrt{1+\sqrt{n}\epsilon_2}],$$

where $|\hat{\lambda}|_{max} = \max(|\hat{\lambda}_1|, |\hat{\lambda}_n|)$.

Proof: Let

$$\begin{aligned} E_1 &= \frac{1}{|\hat{\lambda}|_{max}}(T\hat{U} - \hat{U}\hat{\Lambda}), & (2.1)\\ E_2 &= \hat{U}^T\hat{U} - I, \text{ and} & (2.2)\\ \bar{E}_2 &= \hat{U}\hat{U}^T - I \end{aligned}$$

By Lemma 2.2.1,

$$\begin{aligned} \| E_1 \|_2 \le \sqrt{n} \max_i \| E_1 e_i \|_2 = \sqrt{n}\mathcal{R} \le \sqrt{n}\epsilon_1, \\ \| E_2 \|_2 \le \sqrt{n} \ \| E_2 \|_\infty = \sqrt{n}\mathcal{O} \le \sqrt{n}\epsilon_2. \end{aligned}$$

The second inequality bounds $\| \hat{U}^T \|_2^{\,2}$ as follows. By Lemmas 2.2.1 and 2.2.2,

$$\begin{aligned} \sqrt{n}\epsilon_2 &\ge \| \bar{E}_2 \|_2 \\ &= \| \hat{U}\hat{U}^T - I \|_2 \\ &\ge \| \hat{U}\hat{U}^T \|_2 - \| I \|_2 \\ &= \| \hat{U}^T \|_2^{\,2} - 1. \end{aligned}$$

Therefore, $\| \hat{U}^T \|_2^{\,2} \le 1 + \sqrt{n}\epsilon_2$. From equation (2.1),

$$T\hat{U}\hat{U}^T - \hat{U}\hat{\Lambda}\hat{U}^T = |\hat{\lambda}|_{max}E_1\hat{U}^T,$$

so that

$$\begin{aligned} \hat{U}\hat{\Lambda}\hat{U}^T &= T\hat{U}\hat{U}^T - |\hat{\lambda}|_{max}E_1\hat{U}^T \\ &= T(I + \bar{E}_2) - |\hat{\lambda}|_{max}E_1\hat{U}^T \\ &= T + E, \end{aligned}$$

where $E = T\bar{E}_2 - |\hat{\lambda}|_{max} E_1 \hat{U}^T$, and

$$\begin{aligned} \| E \|_2 &\leq \left[\, |\lambda|_{max} \| \bar{E}_2 \|_2 + |\hat{\lambda}|_{max} \| E_1 \|_2 \| \hat{U}^T \|_2 \, \right] \\ &\leq \sqrt{n}[\, |\lambda|_{max}\epsilon_2 + |\hat{\lambda}|_{max}\epsilon_1 \sqrt{1 + \sqrt{n}\epsilon_2}]. \end{aligned}$$

■

Note that if the bounds

$$\| T\hat{U} - \hat{U}\hat{\Lambda} \|_2 \leq \epsilon_1$$

$$\| \hat{U}^T\hat{U} - I \|_2 \leq \epsilon_2$$

are satisfied, the error matrix is bounded above by

$$\| E \|_2 \;\leq\; |\lambda|_{max}\epsilon_2 + |\hat{\lambda}|_{max}\epsilon_1\sqrt{1 + \epsilon_2},$$

which is independent of the matrix order. The infinity norm and the maximum column 2-norm are used in practice because they are less expensive to compute.

The result of Theorem 2.2.1 is extended to the SVD of B in Theorem 2.2.2 which also shows that if $\mathcal{R}_X$, $\mathcal{R}_Y$, $\mathcal{O}_X$, and $\mathcal{O}_Y$ are small, $\hat{Y}\hat{\Sigma}\hat{X}^T$ is the exact SVD of a matrix nearly equal to B. Theorem 2.2.2 shows that it suffices to compute one residual when determining the accuracy of the residual.

Theorem 2.2.2 *Let $\hat{Y}\hat{\Sigma}\hat{X}^T$ be the computed singular value decomposition of a bidiagonal matrix B. If $\mathcal{R}_X \leq \epsilon_1$, $\mathcal{R}_Y \leq \epsilon_2$, $\mathcal{O}_X \leq \epsilon_3$, and $\mathcal{O}_Y \leq \epsilon_4$, then*

$$B = \hat{Y}\hat{\Sigma}\hat{X}^T + E,$$

where $\| E \|_2 \leq \sqrt{n}[\ \min(\sigma_1\epsilon_3 + \epsilon_1\hat{\sigma}_1\sqrt{1 + \sqrt{n}\epsilon_3},\ \sigma_1\epsilon_4 + \epsilon_2\hat{\sigma}_1\sqrt{1 + \sqrt{n}\epsilon_4}\ ]$. *In addition, up to first order*

$$\| \hat{Y}^T B - \hat{\Sigma}\hat{X}^T \|_2 \leq \sqrt{n}(\hat{\sigma}_1\epsilon_1 + \sigma_1\epsilon_3 + \hat{\sigma}_4).$$

Proof: Let

$$\begin{aligned}
E_1 &= \frac{1}{\hat{\sigma}_1}(B\hat{X} - \hat{Y}\hat{\Sigma}), && (2.3)\\
E_2 &= \frac{1}{\hat{\sigma}_1}(\hat{Y}^T B - \hat{\Sigma}\hat{X}^T), && (2.4)\\
E_3 &= \hat{X}^T\hat{X} - I, && (2.5)\\
\bar{E}_3 &= \hat{X}\hat{X}^T - I, && (2.6)\\
E_4 &= \hat{Y}\hat{Y}^T - I, \textit{ and} && (2.7)\\
\bar{E}_4 &= \hat{Y}^T\hat{Y} - I,. && (2.8)
\end{aligned}$$

By Lemmas 2.2.1 – 2.2.2,

$$\begin{aligned}
\sqrt{n}\epsilon_3 &\geq \| \bar{E}_3 \|_2 \\
&= \| \hat{X}\hat{X}^T - I \|_2 \\
&\geq \| \hat{X}\hat{X}^T \|_2 - \| I \|_2 \\
&= \| \hat{X}^T \|_2^{\,2} - 1.
\end{aligned}$$

Therefore, $\| \hat{X}^T \|_2^{\,2} \leq 1 + \sqrt{n}\epsilon_3$. From equation (2.3),

$$B\hat{X}\hat{X}^T - \hat{Y}\hat{\Sigma}\hat{X}^T = \hat{\sigma}_1 E_1 \hat{X}^T,$$

so that, by equation (2.6),

$$\begin{aligned}
\hat{Y}\hat{\Sigma}\hat{X}^T &= B\hat{X}\hat{X}^T - \hat{\sigma}_1 E_1 \hat{X}^T \\
&= B(I + \bar{E}_3) - \hat{\sigma}_1 E_1 \hat{X}^T \\
&= B + E,
\end{aligned}$$

where $E = B\bar{E}_3 - \hat{\sigma}_1 E_1 \hat{X}^T$. Because $\| B \|_2 = \sigma_1$,

$$\begin{aligned}
\| E \|_2 &\leq \left[\sigma_1 \| \bar{E}_3 \|_2 + \hat{\sigma}_1 \| E_1 \|_2 \| \hat{X}^T \|_2 \right] \\
&\leq \sqrt{n}[\sigma_1\epsilon_3 + \hat{\sigma}_1\epsilon_1\sqrt{1 + \sqrt{n}\epsilon_3}].
\end{aligned}$$

In the same way, equations (2.4) and (2.7) show that

$$\begin{aligned}
\| E \|_2 &\leq \left[\sigma_1 \| \bar{E}_4 \|_2 + \hat{\sigma}_1 \| E_2 \|_2 \| \hat{Y}^T \|_2 \right] \\
&\leq \sqrt{n}[\sigma_1\epsilon_4 + \hat{\sigma}_1\epsilon_2\sqrt{1 + \sqrt{n}\epsilon_4}].
\end{aligned}$$

Postmultiplying equation (2.3) by $\hat{\sigma}_1 \hat{X}^T$ then premultiplying it by Y^T gives

$$\hat{Y}^T B \hat{X} \hat{X}^T - \hat{Y}^T \hat{Y} \hat{\Sigma} \hat{X}^T = \hat{\sigma}_1 \hat{Y}^T E_1 \hat{X}^T.$$

Because $\| \hat{X}^T \hat{X} - I \|_2 = \| \hat{X} \hat{X}^T - I \|_2$,

$$\hat{X} \hat{X}^T - I = \bar{E}_3 \text{ with } \| \bar{E}_3 \|_2 \leq \epsilon_3,$$

and

$$\hat{Y}^T B(I + \bar{E}_3) - (I + E_4) \hat{\Sigma} \hat{X}^T = \hat{\sigma}_1 \hat{Y}^T E_1 \hat{X}^T.$$

Rearranging the terms,

$$\hat{Y}^T B - \hat{\Sigma} \hat{X}^T = \hat{\sigma}_1 \hat{Y}^T E_1 \hat{X}^T - \hat{Y}^T B \bar{E}_3 + E_4 \hat{\Sigma} \hat{X}^T.$$

Because $\| \hat{\Sigma} \|_2 = \hat{\sigma}_1$,

$$\| \hat{Y}^T B - \hat{\Sigma} \hat{X}^T \|_2 \leq \sqrt{n} (\hat{\sigma}_1 \epsilon_1 \| \hat{Y}^T \|_2 \| \hat{X}^T \|_2 + \sigma_1 \epsilon_3 \| \hat{Y}^T \|_2 + \epsilon_4 \hat{\sigma}_1 \| \hat{X}^T \|_2).$$

As in the proof of Theorem 2.2.1, $\| \hat{X}^T \|_2 \leq \sqrt{1 + \sqrt{n} \epsilon_3}$ and $\| \hat{Y}^T \|_2 \leq \sqrt{1 + \sqrt{n} \epsilon_4}$, so that

$$\| \hat{Y}^T B - \hat{\Sigma} \hat{X}^T \|_2 \leq \sqrt{n} (\hat{\sigma}_1 \epsilon_1 + \sigma_1 \epsilon_3 + \hat{\sigma}_1 \epsilon_4) + \text{higher order terms.}$$

When $\sqrt{n} \epsilon_j << 1$ for $j = 1, \ldots, 4$, the higher order terms are a small contribution to the total bound, and for accurately computed singular values $\hat{\sigma}_1 \approx \sigma_1$.

Only the above norm-based quality measures are used in this thesis. Other quality measures used when T and B are known to very high accuracy are discussed in [3, 15, 18, 19].

2.3 Test Matrices

The serial and parallel eigensolvers were tested on the collection of matrices given in this section. The matrices with clustered eigenvalues or singular values or small singular values present difficult problems. A pair of computed eigenvalues $\hat{\lambda}_i$, $\hat{\lambda}_{i+1}$ belong to a *cluster* if $\hat{\lambda}_i - \hat{\lambda}_{i+1} \leq 10^{-14} |\hat{\lambda}|_{max}$.

2.3.1 Tridiagonal Test Matrices

1. Matrix [1,2,1]: The Toeplitz matrix, $T = [1, 2, 1]$ has eigenvalues given by

$$\lambda_j = 2\left(1 + \cos\frac{j\pi}{n+1}\right),$$

   for $j = 1, \ldots, n$ [36]. The eigenvalues are more closely spaced at the ends of the spectrum than in the center, and the spacing decreases with increasing matrix order. For the matrix orders tested, all eigenvalues are computationally distinct. Matrices of this form arise, for example, in the solution of boundary value problems by difference methods [72].

2. Modified matrix [1,2,1]: This matrix is formed from matrix [1,2,1] by setting the fifth through eighth off-diagonal elements $\beta_5, \ldots, \beta_8$ equal to 10^{-14}. This contrived example has several clustered eigenvalues and so presents a difficult eigenproblem.

3. Matrix [1,u,1]: The matrix [1,u,1] has the value 1 in each off-diagonal position and the value $i \times 10^{-6}$ in the ith diagonal position. This matrix undergoes little deflation when its eigendecomposition is computed by Cuppen's divide and conquer method [13].

4. W_n^+: The Wilkinson matrix of odd order n [76] has diagonal elements $\lfloor\frac{n}{2}\rfloor, \ldots, 1, 0, 1, \ldots, \lfloor\frac{n}{2}\rfloor$ and all off-diagonal elements equal to 1 [76]. The spacing of the eigenvalues decreases with increasing eigenvalues until the largest pairs (computed in double precision) are computationally coincident for orders of about 13 and larger.

5. W_g^+: The glued Wilkinson matrix of order $21j$ is formed by placing j copies of W_{21}^+ along the diagonal of the matrix and setting off-diagonal elements equal to 10^{-14} at the positions $\beta_{21}, \beta_{42}, \ldots$ where the submatrices join. For matrix orders greater than about 200, W_g^+ has clusters of eigenvalues near the integers $1, 2, \ldots, \lfloor\frac{n}{2}\rfloor$ [58].

order	largest cluster size	number of clustered eigenvalues	number of test matrices
100	5	$0 - 26$	49
100	51	51	1
500	8	35-83	5

Table 2.1: Characteristics of the test matrices with clustered eigenvalues.

6. Random: These matrices have uniformly distributed pseudorandom entries between -1 and 1 as both diagonal and off-diagonal elements. The random elements are generated using the linear congruential random number generator RAND available from NETLIB. It turns out that the matrices generated for the experiments have minimum eigenvalue spacing 10^{-4} for orders through 525.

7. Matrices with clustered eigenvalues: These matrices are formed from diagonal matrices with some repeated eigenvalues by applying nearly orthogonal transformations. Specifically, the matrices are generated by multiplying $\hat{U}D\hat{U}^T$ where D is a diagonal matrix and $\hat{U}$ is the matrix of eigenvectors of matrix $[1, 2, 1]$ computed using the EISPACK QL routine TQL2 [68]. The product $\hat{U}D\hat{U}^T$ was reduced to tridiagonal form by the EISPACK routine TRED2. Table 2.1 categorizes the test matrices by the number of eigenvalues in the largest cluster. All of the test matrices have norms $\| T \|_2 \approx 1$ except for six of order 500 which have $\| T \|_2 \approx 50$.

Matrix [1,2,1] and the modified matrix [1,2,1] are positive definite; the rest are indefinite.

2.3.2 Bidiagonal Test Matrices

1. Matrix [2,1]: All singular values lie within the interval [1,3]. For the matrix orders tested, all singular values are computationally distinct.

2. Random: These matrices have uniformly distributed random entries between -1 and 1 generated by RAND on both diagonal and off-diagonal elements. The matrices tested turn out to have well-separated singular values with minimum magnitude $O(10^{-5})$.

3. B_W: Inspired by the Wilkinson matrix W^+, this matrix of even order has diagonal elements $\frac{n}{2}, \ldots, 1, 1, \ldots, \frac{n}{2}$ and all off-diagonal elements equal to 1. Its smallest singular value is $O(10^{-3})$, and its largest ones appear in computationally coincident pairs for orders of about ten and larger in double precision.

4. Matrix $[2, u]/n$: The matrix $[2, u]/n$ of order n has the value $2/n$ in each diagonal position and the value i/n in the ith off-diagonal position. This matrix has one singular value less than 10^{-14} for orders greater than eighty.

5. Modified matrix [2,1]: This matrix is formed from matrix [2,1] by setting the sixth through ninth diagonal elements $\alpha_6, \ldots, \alpha_9$ and fifth through eighth off-diagonal elements $\beta_5, \ldots, \beta_8$ equal to 10^{-14}. This matrix is ill-conditioned, having between four and eight singular values less than 10^{-8} and between two and four singular values less than 10^{-14} for all tested orders.

Chapter 3

The Hypercube Multiprocessor

This chapter describes the distributed-memory hypercube multiprocessor used for the parallel experiments presented in this thesis. Section 3.1 introduces the hypercube graph defining the architecture of the multiprocessor, and Section 3.2 describes the Intel iPSC/1-d5M used for the parallel experiments. Section 3.3 gives the algorithms for data transmission, matrix multiplication, and orthogonalization of vectors used in the eigensolvers examined in Chapter 5.

3.1 Characterization of the Hypercube

A hypercube of dimension $d \geq 0$, or d-cube, is a graph consisting of $p = 2^d$ nodes and is defined recursively as follows. A 0-cube is composed of a single node, while for $d > 0$, a d-cube is obtained by adding edges between corresponding nodes of two $(d-1)$-cubes. (See Figure 3.1.) This construction shows that a 3-cube is formed of two 2-cubes, four 1-cubes, or eight 0-cubes. In general, a d-cube is made up of 2^{d-j} j-cubes, for $0 \leq j < d$. Alternatively, a d-cube may be defined by associating with each of the 2^d nodes a binary label of length d so that every edge connects two nodes whose labels differ in exactly one bit. In a d-cube, every node is connected to d others making a total of $d2^{d-1}$ edges.

The nodes of a hypercube may be ordered according to a binary reflected Gray code [29, 61] so that successive nodes in the ordering are physically adjacent. Because a Gray code sequence is cyclic, it defines a *ring* of physically

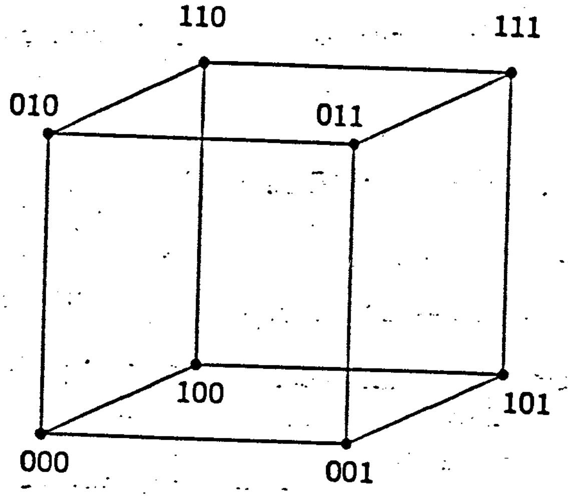

Figure 3.1: A 3-cube with numbered nodes

interconnected processors within the hypercube. Within the ring, node j has two neighbors whose binary identifiers differ from its own in a single bit. For example, a ring is *embedded* in the hypercube of Figure 3.1 by passing from one node to the next in the order 000, 001, 011, 010, 110, 111, 101, 100, 000. Embeddings into the hypercube of toroids or one- and two-dimensional arrays are also based on Gray codes [63].

3.2 The Multiprocessor

A hypercube *multiprocessor* of dimension d is made up of $p = 2^d$ processors located at the nodes of a d-dimensional hypercube graph and $d2^{d-1}$ processor interconnections corresponding to the edges of the graph. The term *hypercube* refers to both graph and multiprocessor.

The parallel eigenvalue codes were implemented on an Intel iPSC/1-d5M hypercube multiprocessor. This machine consists of 32 identical node processors, each capable of communicating directly with five neighboring processors. Each processor has direct access to its own local memory only and exchanges data with other processors through message passing. A node can communicate with only one of its neighbors at a time and does so by issuing a *send* communication primitive to initiate a message transfer or a *receive* primitive to accept a message sent to it by another processor. Messages arriving at a node are held in a queue until selected via a receive command. Each processor has 4.5 Mbytes of local memory.

A separate processor serves as the cube manager or host machine. It can communicate with all nodes via a global bus. For the implementations discussed in this thesis, the host is used for downloading code onto, passing initial data to, and accumulating final results from the node processors but does not enter into the computation in any other way. In the remainder of this thesis, a node processor is named by the decimal value of its binary identifier, while the cube manager is known as the host.

For purposes of estimating computation times on the hypercube, it is assumed that time $\beta + k\tau$ is required to send a double precision vector of length k from one processor to a neighbor, where β is the communication startup time and τ is the time to transfer one vector element. Using the terminology of [33], the time required to perform a floating point operation (*flop*) of the form

$$c_{ij} = c_{ij} + a_{ik}b_{kj}$$

is denoted by ω; it includes the time for a floating point multiplication and addition as well as for some pointer manipulation. If the array elements are real double precision floating point numbers, then $\frac{\beta}{\omega} \approx 10$ and $\frac{\beta}{\tau} \approx 125$ on the iPSC/1-d5M running operating system release R3.0. These values were derived by timing the double precision LINPACK benchmark on a single processor (for ω) and by timing messages sent around a ring of 32 processors (for β and τ).

3.3 Some Hypercube Algorithms

In general, an algorithm is implemented in parallel by dividing the work required into parts or *tasks*, some or all of which can be executed simultaneously. In the implementations to be discussed, tasks are assigned *statically* to the processors without the use of a task queue. That is, tasks are assigned to processors *a priori* as a function of the matrix order and the number of processors. The details of these assignments differ for each method and are discussed in Chapter 5. This strategy permits simplicity of programming and reduction of scheduling overhead, although it does not necessarily provide the best processor load balance.

In the implementations of hypercube algorithms described in this thesis, matrices are stored by assigning an entire column or an entire row to one processor. For simplicity, it is assumed that the number of processors p divides the order n of the matrix. In the actual implementation, the rows or columns are assigned to processors in such a way that no processor contains more than $\lceil n/p \rceil$ of them. In all descriptions, the processors are labelled P_j, $0 \leq j \leq p-1$. The processor indices should always be taken modulo p.

3.3.1 Data Transmission on the Hypercube

The *Alternate Direction Exchange Algorithm (ADE)* [64] uses the recursive structure of the hypercube to carry out a total exchange of the data held by the processors of a cube. A vector of length k is broadcast from each processor to all others in a d-cube in d data transmission steps where the amount of data doubles during each step. At step l, processors separate into two $(d-1)$-cubes S_0 and S_1 according to the value of bit l in their binary labels. The binary identifier of processor P_j is denoted β_j, and P_j begins with a vector $v_{j+1}^{(1)}$ of length k.

Algorithm 3.3.1 (Alternate Direction Exchange)

```
In parallel, do on all processors $P_j$ with binary labels $\beta_j$, $0 \leq j \leq 2^d - 1$:
  For $l = 1, \ldots, d$:
      1. Pair with processor $P_{j'}$ whose binary label differs from $\beta_j$ only in bit $l$.
      2. Send vector $v_{j+1}^{(l)}$ of length $2^{l-1}k$ to processor $P_{j'}$.
```

3. Receive vector $v_{j'+1}^{(l)}$ of length $2^{l-1}k$ from processor $P_{j'}$.

4. Concatenate $v_{j+1}^{(l)}$ with $v_{j'+1}^{(l)}$ to form the vector $v_{j+1}^{(l+1)}$ of length $2^l k$.

The time to perform Algorithm 3.3.1 (ADE) on a d-cube is

$$\begin{aligned}\tau_{ADE}^{d} &= 2[(\beta + k\tau) + (\beta + 2k\tau) + \cdots + (\beta + 2^{d-1}k\tau)] \\ &= 2[d\beta + (2^d - 1)k\tau].\end{aligned}$$

The factor of two reflects the fact that messages can only be sent in one direction at a time on a node-to-node link.

The above algorithm can also be used when data are broadcast within a subcube S of dimension $j \leq d$. S is then made up of all processors of the d-cube whose binary labels agree in exactly the same $d - j$ bit positions. The corresponding communication time is

$$\tau_{ADE}^{j} = 2[j\beta + (2^j - 1)k\tau].$$

The 2^{d-j} subcubes of dimension j comprising a d-cube can simultaneously perform an alternate direction exchange without interference.

3.3.2 Matrix Multiplication on the Hypercube

In this section, the processors in a d-cube are ordered according to a binary reflected Gray code sequence [29, 61], and the processor labelled P_j is the jth member in this sequence, $0 \leq j \leq 2^d - 1$.

When using an embedded ring, a matrix may be stored in the hypercube by situating blocks of adjacent columns in neighboring processors. Ring Matrix Multiplication (RMM) described in Algorithm 3.3.2 multiplies two $n \times n$ matrices A and B both distributed by block columns among the processors of a d-cube, where $n = k2^d$.

Initially, processor P_j, $0 \leq j \leq 2^d - 1$, contains columns $jk + 1, \ldots, (j + 1)k$ of A and of B and, upon completion, columns $jk + 1, \ldots, (j + 1)k$ of C. During the formation of C, the columns of B remain in their original places while the columns of A are passed around the ring from processor to processor, overwriting the previously-held columns of A in each processor.

Let $\tilde{B}_{ij}$ denote the $k \times k$ block matrix with its first element in position (ik, jk) of B, $1 \leq i, j \leq 2^d$, and let the block vectors

$$\tilde{B}_j = [\tilde{B}_{1j}^T \ldots \tilde{B}_{2^d,j}^T]^T$$

be the k columns $jk+1, \ldots, (j+1)k$ of B. Similarly, the block vectors $\tilde{A}_j$ and $\tilde{C}_j$ comprise the k columns $jk+1, \ldots, (j+1)k$ of A and C, respectively. $\tilde{S}_j$ is the k column block vector used to accumulate the sum.

Algorithm 3.3.2 (Ring Matrix Multiplication)

```
In parallel, do on all processors $P_j$, $0 \leq j \leq 2^d - 1$:
  $\tilde{S}_{j+1} = 0$
  For $i = 1, \ldots, 2^d$:
    1. Compute $\tilde{S}_{j+1} = \tilde{S}_{j+1} + \tilde{A}_{j-i+2}\tilde{B}_{j-i+2,j+1}$
    2. Send $\tilde{A}_{j-i+2}$ to processor $P_{j+1}$
    3. Receive $\tilde{A}_{j-i+1}$ from processor $P_{j-1}$
$\tilde{C}_{j+1} = \tilde{S}_{j+1}$
```

■

The arithmetic time for each iteration is $2^d k^3 \omega$, and the communication time is $2(\beta + 2^d k^2 \tau)$, giving a total time of

$$\tau_{RMM}^d = 2^{2d} k^3 \omega + 2^{d+1}(\beta + 2^d k^2 \tau).$$

The algorithm was designed to take advantage of the large memory of the iPSC/1-d5M. That is, it assumes buffers large enough to accommodate the incoming block columns. Alternative algorithms are described in [28].

3.3.3 Modified Gram-Schmidt Orthogonalization

The *Modified Gram-Schmidt (MGS)* procedure transforms a set of linearly independent vectors into a set of orthonormal vectors. This is necessary, for example, when inverse iteration is applied to poorly separated eigenvalues and produces eigenvectors that, while linearly independent, are not orthogonal [76].

Algorithm 3.3.3 overwrites a set of m linearly independent vectors $\{v_1, \ldots, v_m\}$ of length n with an orthonormal set $\{\hat{v}_1, \ldots, \hat{v}_m\}$ spanning the same space. It is assumed that $1 < m \leq p$ and that v_{j+1} resides in processor P_j, the jth processor in an embedded linear array. The kth orthonormalized vector is computed during the kth step of MGS. The orthonormalized vectors are passed from processor to processor to effect the orthonormalization of the remaining vectors.

Algorithm 3.3.3 (Modified Gram-Schmidt Orthogonalization ($m \leq p$))

```
In parallel, do on all processors $P_j$, $0 \leq j \leq m-1$:
  1. For $k = 1, \ldots, m$
      (a) if $j > k$, receive kth vector $\hat{v}_k = (\hat{v}_{1k}, \ldots, \hat{v}_{nk})^T$ from processor $P_{j-1}$
      (b) if $j < m-1$, send $\hat{v}_k$ to processor $P_{j+1}$
      (c) compute $r_{k,j+1} = \hat{v}_k^T v_{j+1}$
      (d) compute $v_{j+1} = v_{j+1} - \hat{v}_k r_{k,j+1}$
  2. Normalize jth vector: $\hat{v}_{j+1} = v_{j+1}/\| v_{j+1} \|_2$
  3. If $j < m-1$, send $\hat{v}_{j+1}$ to processor $P_{j+1}$
```

■

Steps 1.a, 1.b, and 1.d can be pipelined, but when $m \leq p$ each processor can spend some idle waiting time during the computation. In general, P_j is idle for the time required to normalize $\hat{v}_1$ and pass it through the j communication links from P_0, and the total time for Algorithm 3.3.3 is the time needed by the processor (P_{m-1}) holding the last vector (v_m). As P_{m-1} need forward neither $\hat{v}_{m-1}$ nor $\hat{v}_m$, this total is the time needed for $\hat{v}_{m-1}$ to arrive at P_{m-1} plus the cost of orthogonalizing v_m with respect to $\hat{v}_{m-1}$ and normalizing v_m.

The time for $\hat{v}_{m-1}$ to arrive at P_{m-1} is determined by the relative costs of computation and communication. When $\beta + n\tau$ (the cost to send a vector from one processor to its neighbor) is large in comparison to $2n\omega$ (the cost of orthogonalizing one vector with respect to another or of normalizing one vector), P_j can be ready to use v_{j-1} as soon as it arrives. In this case, P_0 normalizes and sends $\hat{v}_1$ in time $2n\omega + (\beta + n\tau)$. For $2 < j < m-1$, P_j forwards $\hat{v}_j$ to P_{j+1}, orthogonalizes v_{j+1} with respect to v_j, and normalizes and forwards v_{j+1} in total time $4n\omega + 2(\beta + n\tau)$. P_{m-1} orthogonalizes v_m with respect to $\hat{v}_{m-1}$ and normalizes the result. An upper bound on the total time is then

$$\tau_{MGS}^{(1)} = (2m-1)2n\omega + (2m-3)(\beta + n\tau).$$

Note that P_j can be idle while awaiting $\hat{v}_2, \ldots, \hat{v}_j$.

When communication is much faster than computation, processor P_{m-2} stays busy once it has begun computation. Because it requires $\hat{v}_1, \ldots, \hat{v}_{m-2}$ for its computation, P_{m-2} requires more time than processors $P_1, \ldots, P_{m-3}$. Assuming that time $\beta + n\tau$ is needed to receive a vector, P_{m-2} takes time $(m-2)(\beta + n\tau)$ to receive $\hat{v}_1$, time $\beta + n\tau$ to forward it to P_{m-1}, and time $2n\omega$ to use $\hat{v}_1$. For $1 < j < m-1$, P_{m-1} takes time $2n\omega + 2(\beta + n\tau)$ to receive and forward vector $\hat{v}_j$. Upon receipt of v_{m-1}, P_{m-1} orthogonalizes v_m with respect to that vector and normalizes the result. An upper bound on the total time is

$$\tau_{MGS}^{(2)} = (m+2)2n\omega + (3m-6)(\beta + n\tau).$$

An upper bound on the time for Algorithm 3.3.3 is thus

$$\max(\tau_M^{(1)}GS, \tau_M^{(2)}GS).$$

Chapter 4

Methods for the Symmetric Tridiagonal Eigenproblem

4.1 Introduction

Cuppen's divide and conquer method, bisection with inverse iteration, and the QL method are the most promising candidates for accurate and efficient parallel solution of the symmetric tridiagonal eigenproblem. This chapter reviews existing algorithms for these methods. The divide and conquer method is described in Section 4.2, bisection with inverse iteration in Section 4.3, and the shifted QL method in Section 4.4. The latter covers both Wilkinson's shift and the perfect shift (*i.e.*, computed eigenvalues used as shifts. Because the perfect shift method has not been studied extensively, Section 4.4 also includes a brief experimental comparison of the two shift strategies.

4.2 The Divide and Conquer Method

Cuppen's divide and conquer method is based on the fact that a symmetric tridiagonal matrix T of order $n = 2m$ can be divided into a pair of equal-sized symmetric tridiagonal submatrices plus a rank one correction

$$T = \begin{pmatrix} T_0 & \\ & T_1 \end{pmatrix} + \theta\beta \begin{pmatrix} e_m \\ \theta^{-1} e_1 \end{pmatrix} \left(e_m^T \quad \theta^{-1} e_1^T \right), \tag{4.1}$$

where β is the mth off-diagonal element of T, e_i is the ith unit vector of length m, and T_0 and T_1 are symmetric tridiagonal of order m. The sign of θ is selected to ensure that subdivision of the matrix does not result in cancellation [24]. The original problem has now been split into two eigenproblems of half its order.

If the solutions to the two smaller eigensystems are $T_0 = X_0 D_0 X_0^T$ and $T_1 = X_1 D_1 X_1^T$, then

$$T = Q\left[D + \beta\theta \begin{pmatrix} l_0 \\ \theta^{-1} f_1 \end{pmatrix} \begin{pmatrix} l_0^T & \theta^{-1} f_1^T \end{pmatrix}\right] Q^T$$

where

$$Q = \begin{pmatrix} X_0 & \\ & X_1 \end{pmatrix}, \quad D = \begin{pmatrix} D_0 & \\ & D_1 \end{pmatrix},$$

$l_0^T = e_m^T X_0$ is the last row of X_0, and $f_1^T = e_1^T X_1$ is the first row of X_1. To solve the eigenproblem for T, it is necessary to find the eigenvalues and eigenvectors of the diagonal plus rank-one matrix

$$D + \rho z z^T = Q^T T Q,$$

where $z^T = \sqrt{\frac{\beta\theta}{\rho}} \begin{pmatrix} l_0^T & \theta^{-1} f_1^T \end{pmatrix}$, and ρ is selected so that $\| z \|_2 = 1$.

The eigensystem of T is computed via the rank-one updating technique described in [31]. Namely, if all elements of z are non-zero and if the diagonal elements of D are distinct, then the eigenvalues of $D + \rho z z^T$ are the roots $\lambda_1 > \ldots > \lambda_n$ of the *secular equation* [31]

$$w(\lambda) = 1 + \rho z^T (D - \lambda)^{-1} z. \tag{4.2}$$

If $\beta > 0$ and the diagonal elements of D are given by $\delta_1 > \ldots > \delta_n$, the eigenvalue is bracketed by adjacent diagonal elements of D ($\delta_{i-1} > \lambda_i > \delta_i$) and $\delta_1 + \rho z^T z > \lambda_1 > \delta_1$. This property means that the roots of $w(\lambda)$ may be found efficiently using rational interpolation [11]. The secular equation is the sum

$$w(\lambda) = 1 + \rho \sum_{i=1}^{n} \frac{\zeta_i^2}{\delta_i - \lambda},$$

where $z = (\zeta_1, \ldots, \zeta_n)^T$. To compute the root λ_j in exact arithmetic, the sum is split into

$$w(\lambda) = 1 + \phi(\lambda) + \psi(\lambda),$$

with

$$\phi(\lambda) = \rho \sum_{i=1}^{j-1} \frac{\zeta_i^2}{\delta_i - \lambda}, \quad \psi(\lambda) = 1 + \rho \sum_{i=j}^{n} \frac{\zeta_i^2}{\delta_i - \lambda},$$

Because $\lambda_j \in (\delta_{j-1}, \delta_j)$, all terms of $\phi(\lambda)$ are positive, and all terms of $\psi(\lambda)$ are negative. Given an initial guess $\gamma_0 \in (\delta_j, \lambda_j)$, $\psi(\lambda)$ and $\phi(\lambda)$ are approximated by the the rational interpolants

$$\frac{p}{q - \lambda}, \quad r + \frac{s}{\delta - \lambda},$$

where $\delta \equiv \lambda_{j+1}$ and

$$\begin{aligned} \frac{p}{q - \gamma_0} &= \psi(\gamma_0), & r + \frac{s}{\delta - \gamma_0} &= \phi(\gamma_0), \\ \frac{p}{(q - \gamma_0)^2} &= \psi'(\gamma_0), & \frac{s}{(\delta - \gamma_0)^2} &= \phi'(\gamma_0). \end{aligned}$$

The first iterate is the solution $\lambda = \gamma_1$ to

$$\frac{-p}{q - \lambda} = 1 + r + \frac{s}{\delta - \lambda}. \tag{4.3}$$

The iteration proceeds by replacing γ_0 with γ_1 above and solving equation (4.3) for $\lambda = \gamma_2$, then replacing γ_1 with γ_2 and solving for $\lambda = \gamma_3$, and so on. The sequence $\gamma_0, \gamma_1, \ldots$ converges to λ_j quadratically and monotonically from one side of the root, thus ensuring that λ_j can be extracted from the interval (δ_j, δ_{j+1}) without need for safeguarding [11]. When $\beta < 0$, a change of variables allows a similar derivation.

Once λ_j has been found, its corresponding eigenvector is computed from

$$u_j = \frac{(D - \lambda_j)^{-1} z}{\| (D - \lambda_j)^{-1} z \|_2}. \tag{4.4}$$

If $U = (u_1, \ldots, u_n)$ and $\Lambda = diag(\lambda_1, \ldots, \lambda_n)$, then the eigendecomposition of the original matrix may be expressed as the product

$$T = QU\Lambda U^T Q^T.$$

The columns of QU are the eigenvectors of T, and the diagonal elements of Λ are its eigenvalues.

The above description depends on having distinct elements along the diagonal of D. In many instances, however, multiplicities do occur. For example, if

$$T = \begin{pmatrix} 2 & 1 & 0 & 0 \\ 1 & 3 & 1 & 0 \\ 0 & 1 & 3 & 1 \\ 0 & 0 & 1 & 2 \end{pmatrix},$$

then $\beta = 1$, and $T_0 = T_1 = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$. Both T_0 and T_1 have eigenvalues 1 and 3, so $D = \begin{pmatrix} 3 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$, and $\delta_1 = \delta_2, \delta_3 = \delta_4$.

When the diagonal elements of D are not distinct, *i.e.*, $\delta_l = \delta_{l+1} = \ldots = \delta_{l+k}$, the eigenproblem of order n is reduced to one of order $n - k$ by a process known as *deflation.* The eigenvector basis is first rotated to zero out the elements $\zeta_{l+1}, \ldots, \zeta_{l+k}$ corresponding to the multiple elements $\delta_{l+1} = \cdots = \delta_{l+k}$: a product of plane rotations G_l is applied so that

$$G_l(\zeta_l, \zeta_{l+1}, \cdots, \zeta_{l+k})^T = (\zeta_l', \zeta_{l+1}', \cdots, \zeta_{l+k}')^T = (\zeta_l', 0, \cdots, 0)^T.$$

For $l + 1 \leq j \leq l + k$, the jth eigenvalue in exact arithmetic is the jth element of D ($\lambda_j = \delta_j$) and its corresponding eigenvector may be chosen as the appropriate unit vector ($u_j = e_j$) [11]. Therefore, multiple values along the diagonal of D result in a significant reduction in the work required to compute the eigensystem of $D + \rho zz^T$. Zero elements of z corresponding to distinct elements of D lead to similar savings and no rotations need be applied.

The divide and conquer technique thus proceeds by deflating the problem and computing the remaining eigenpairs. Representing the product of all rotations by the matrix G, the matrix T is expressed as

$$T = QG^TU\Lambda U^TGQ^T = X\Lambda X^T, \tag{4.5}$$

where $U\Lambda U^T$ is the eigendecomposition of $G(D+\rho zz^T)G^T$. The eigenvalues of T are the diagonal elements of Λ, and the eigenvectors of T are the columns of $X = QG^TU$.

The above derivation assumes exact arithmetic. Deflation rules have also been developed for finite precision in [24]: rotations are applied when diagonal elements of D are close, and deflation occurs when elements of z are small. Let $\delta_i - \delta_{i+1} = \epsilon$, then consider the 2×2 submatrix of $D + \rho zz^T$

$$\begin{pmatrix} \delta_i & \\ & \delta_{i+1} \end{pmatrix} + \begin{pmatrix} \zeta_i \\ \zeta_{i+1} \end{pmatrix} \begin{pmatrix} \zeta_i, & \zeta_{i+1} \end{pmatrix}.$$

As in the finite precison case, a Givens rotation is devised to reduce ζ_{i+1} to zero

$$\begin{pmatrix} \gamma & \sigma \\ -\sigma & \gamma \end{pmatrix} \left[\begin{pmatrix} \delta_i & \\ & \delta_{i+1} \end{pmatrix} + \begin{pmatrix} \zeta_i \\ \zeta_{i+1} \end{pmatrix} \begin{pmatrix} \zeta_i, & \zeta_{i+1} \end{pmatrix} \right] \begin{pmatrix} \gamma & -\sigma \\ \sigma & \gamma \end{pmatrix}$$

$$= \begin{pmatrix} \delta_i' & \phi_i \\ \phi_i & \delta_{i+1}' \end{pmatrix} + \begin{pmatrix} \zeta_i' \\ 0 \end{pmatrix} \begin{pmatrix} \zeta_i', & 0 \end{pmatrix}, \qquad (4.6)$$

with $\gamma^2 + \sigma^2 = 1$, $(\zeta_i')^2 = \zeta_i^2 + \zeta_{i+1}^2$, and $|\phi_i| = |\gamma\sigma(\delta_{i+1} - \delta_i)|$. The problem is deflated whenever the first matrix on the righthand side of equation (4.6) is diagonal, *i.e.*, whenever $|\phi_i|$ is no larger than an error tolerance which is roughly a unit multiple of $\epsilon_M \| T \|_2$. Numerical experiments have confirmed that the increase in speed due to this deflation is substantial for serial and shared-memory parallel implementations [24]. Note that no existing implementation of the root finder has been proven to provide eigenvalues accurate enough to guarantee orthogonal computed eigenvectors.

4.3 Bisection and Inverse Iteration

Let T be the $n \times n$ symmetric tridiagonal matrix with diagonal elements $\alpha_1, \ldots, \alpha_n$ and off-diagonal elements $\beta_2, \ldots, \beta_n$. By Gerschgorin's Theorem, the n eigenvalues of T lie in the union of the n disks

$$|\lambda - \alpha_i| \le |\beta_i| + |\beta_{i+1}|, \quad 1 \le i \le n.$$

Individual eigenvalues are located in this interval by solving the characteristic equation $\det(T - \lambda) = 0$. The sequence of leading principal minors of the matrix $T - \lambda$ is given by the linear recurrence

$$\begin{aligned} p_0(\lambda) &= 1 \\ p_1(\lambda) &= \alpha_1 - \lambda \\ p_i(\lambda) &= (\alpha_i - \lambda)p_{i-1} - \beta_{i-1}^2 p_{i-2}(\lambda), \ i = 2, \ldots, n. \end{aligned} \tag{4.7}$$

The number of eigenvalues of T less than λ is equal to the number of sign changes in the sequence $\{p_i(\lambda)\}$ [30].

Because the linear recurrence in equation (4.7) is prone to overflow and underflow, it is preferable to use

$$q_i(\lambda) = \frac{p_i(\lambda)}{p_{i-1}(\lambda)}, \ i = 1, \ldots, n. \tag{4.8}$$

The number of eigenvalues less than λ is equal to the number $\gamma(\lambda)$ of negative terms in $\{q_i(\lambda)\}$ [6], and the number of eigenvalues in the interval $[\lambda_1, \lambda_2)$ is given by $\gamma(\lambda_2) - \gamma(\lambda_1)$. (In [51], Kahan shows that this "overflow-free" sequence can sometimes overflow.)

Because $\{p_i(\lambda)\}$ and $\{q_i(\lambda)\}$ are Sturm sequences [42, 76], the eigenvalues of T can be computed by repeated bisection of the initial Gerschgorin interval. Empty intervals are discarded from the search area, and occupied intervals are further bisected until single eigenvalues have been extracted to a given tolerance or until groups of eigenvalues have been confined to within a width smaller than that tolerance. The groups represent *clusters* of computationally coincident eigenvalues. Throughout this thesis, arithmetic is assumed to be monotonic so that $\gamma(\lambda)$ is monotonic [51].

When a single eigenvalue has been isolated within an interval, computation can be accelerated by an interpolation scheme [8] or a faster root-finder such as Zeroin [27]. These enhancements are not considered in this thesis, but they could be used to further improve the performance of parallel bisection.

Once the eigenvalues have been computed by bisection, the corresponding eigenvectors are found with inverse iteration. Inverse iteration for computing u_j is the power method [42] applied with $(T - \hat{\lambda}_j)^{-1}$, where $\hat{\lambda}_1 \geq \ldots \geq \hat{\lambda}_n$ are the computed eigenvalues of T. Using the initial vector z_0, inverse iteration has the simple form [33]

For $l = 1, 2, \ldots$

1. Solve $(T - \hat{\lambda}_j)v_l = z_{l-1}$
2. Normalize $z_l = v_l / \| v_l \|_\infty$.

If $z_0 = \sum_{i=1}^n \alpha_i u_i$, the first iterate has the form $z_1 = \sum_{i=1}^n \frac{\alpha_i}{\lambda_i - \hat{\lambda}_j} u_i$. When $\hat{\lambda}_{j-1}, \lambda_j, \hat{\lambda}_{j+1}$ are computationally distinct and $|\alpha_j|$ is not too small, z_1 has its largest component in the u_j direction, and inverse iteration converges in only a few iterations. When all of the eigenvalues are well-separated and none of the coefficients $|\alpha_i|$ is too small, inverse iteration generates orthogonal eigenvectors [76]. When the eigenvalues are close, the rate of convergence of inverse iteration decreases and the resultant eigenvectors, although independent, are not necessarily orthogonal [76]. Thus, an additional orthogonalization step is needed.

When computationally coincident eigenvalues occur, standard inverse iteration cannot be employed to find the eigenspace corresponding to the multiple eigenvalue because it converges to a single eigenvector [76]. In the EISPACK routine TINVIT [68], computationally coincident eigenvalues are perturbed to a separation ϵ on the order of machine precision times a norm of T as suggested in [76], *i.e.*, coincident computed eigenvalues $\hat{\lambda}_i = \hat{\lambda}_{i+1} = \hat{\lambda}_{i+2}$ are replaced by $\hat{\lambda}_i$, $\hat{\lambda}_{i+1} + \epsilon$, and $\hat{\lambda}_{i+1} + 2\epsilon$ during the computation of eigenvectors.

The routine for finding all eigenvalues of a symmetric, tridiagonal matrix using bisection and the corresponding eigenvectors using inverse iteration is summarized in Algorithm 4.3.1. These steps are the same as used in the EISPACK routine TSTURM or in the combination of EISPACK's BISECT or TRIDIB with TINVIT [68].

Algorithm 4.3.1 (Bisection with Inverse Iteration)

1. *Determine initial search area:*

   Find intervals containing all eigenvalues (e.g., from Gerschgorin disks.)

2. *Compute eigenvalues:*

   Use bisection to determine all eigenvalues.

3. *Compute eigenvectors:*

   Compute the eigenvectors by inverse iteration. Treat eigenvectors corresponding to clustered eigenvalues by appropriately perturbing the eigenvalues in the cluster. After each iteration, use the modified Gram-Schmidt procedure to orthogonalize eigenvectors corresponding to close eigenvalues.

4.4 The QL Method

Any square matrix T can be factored into the form

$$T = QL,$$

where Q is orthogonal and L is lower triangular. The shifted QL method is defined by the following iteration for $k = 0, 1, \dots$ [33]

$$\begin{aligned} T_0 &= T && (4.9) \\ T_k - \mu_k &= Q_k L_k \qquad (k \geq 0) \\ T_{k+1} &= L_k Q_k + \mu_k \\ &= Q_k^T T_k Q_k, && (4.10) \end{aligned}$$

where μ_k is the shift at iteration k. When $T_0 = T$ is symmetric and tridiagonal, each iterate T_k is also symmetric and tridiagonal [9]. As k approaches infinity, the iterates T_k converge to a diagonal matrix with the eigenvalues of T along its diagonal. The columns of the accumulated product of orthogonal transformations $Q_0 \ldots Q_k$ are the eigenvectors of T.

In the *explicit* QL method, the shift μ_k is explicitly subtracted from each diagonal element of T_k. When the elements of T_k have widely varying orders of magnitude, this subtraction can lead to loss of accuracy in the eigenvalues with smallest magnitudes [25]. In that case, it is preferable to use the *implicit* QL algorithm which is mathematically equivalent in exact arithmetic and is intended to prevent loss of accuracy from cancellation. The shift is incorporated into the rotations, and the subtraction $T_k - \mu$ is never explicitly performed [25].

Convergence of the implicit and explicit QL algorithms depends on the shifts used. The EISPACK [68] implementations (TQL2 and IMTQL2, respectively) use the Wilkinson shift. When the QL method is applied to the iterate T_k, the Wilkinson shift is defined as the eigenvalue of its leading 2 × 2 submatrix

$$\begin{pmatrix} \alpha_1^{(k)} & \beta_1^{(k)} \\ \beta_1^{(k)} & \alpha_2^{(k)} \end{pmatrix}$$

closest to $\alpha_1^{(k)}$. The QL method with the Wilkinson shift converges quadratically in exact arithmetic and, in practice, converges cubically in finite precision arithmetic [9].

In general, the leading off-diagonal element converges most quickly to zero although other off-diagonal elements also decrease in magnitude at each iteration [9]. The first diagonal element $\alpha_1^{(k)}$ of T_k is accepted as an eigenvalue of T if the first off-diagonal element $\beta_1^{(k)}$ is negligible. In IMTQL2, the convergence criterion is [68]

$$|\beta_1^{(k)}| < \epsilon_M(|\alpha_1^{(k)}| + |\alpha_2^{(k)}|).$$

The computed eigenvalue $\alpha_1^{(k)}$ is usually, but not always, the eigenvalue of T closest to the shift μ_k [9].

After convergence of one eigenvalue, iteration continues with the order $n-1$ trailing submatrix of T_k. If any other off-diagonal element $\beta_j^{(k)}$, $j > 1$, becomes negligible, the matrix is split at that point and iteration continued with the leading unreduced submatrix [68].

4.4.1 The Perfect-Shift QL Method

An alternative to the Wilkinson shift is to use accurately computed eigenvalues $\hat{\lambda}_1, \ldots, \hat{\lambda}_n$ as shifts as suggested in [59, 62]. The resulting method is called the *ultimate-shift* or *perfect-shift* QL method. The only version of the perfect-shift method considered in this thesis is implemented by altering EISPACK's IMTQL2 to use computed eigenvalues as shifts.

In exact arithmetic, if the shift μ equals an eigenvalue λ_j, the QL method computes the eigenpair (λ_j, u_j) in one iteration [59]. The experimental results in Section 4.4.2 show that with finite precision arithmetic, more than one iteration is generally needed and that the precise number of iterations is strongly dependent on the ordering of the shifts. If eigenvalues are provided as shifts in increasing order, for example, convergence of the method can be guaranteed only if no splitting occurs. If the matrix splits with shift $\mu = \hat{\lambda}_j$ and iteration continues with $\mu = \hat{\lambda}_j$ for the leading unreduced submatrix, then the perfect-shift QL method converges only if $\hat{\lambda}_j$ is an eigenvalue of that submatrix.

As matrix splittings cannot be predicted, it is necessary to devise a means of ensuring that the correct shifts are used at each iteration. One option is to use the QL method using Wilkinson's shift to compute an eigenvalue and then use perfect-shift QL immediately to compute its eigenvector [17]. This strategy works in practice [17] although convergence to the correct eigenvector cannot generally be guaranteed.

Another option is to compute all eigenvalues at once then to compute all eigenvectors. An eigenvalue is used as a shift only if Sturm sequence evaluation shows it to be an eigenvalue of the submatrix at hand. A shift is used until the QL method converges to an eigenvalue or until the matrix splits. After the matrix splits for the first time, Sturm sequence evaluations are used to determine if a computed eigenvalue $\hat{\lambda}_j$ is a valid shift. Assuming that $\hat{\lambda}_1 \geq \ldots \geq \hat{\lambda}_n$, the number of negative terms $\gamma(\lambda)$ in the overflow-free Sturm sequence of equation (4.8) is determined at $\lambda = \frac{1}{2}(\hat{\lambda}_{j-1} + \hat{\lambda}_j)$. If $\gamma(\lambda) = 0$, $\hat{\lambda}_j$ is not an eigenvalue of the submatrix. If $\gamma(\lambda)$ lies between 0 and the order of the submatrix, then $\hat{\lambda}_j$ may be used as a shift. If $\gamma(\lambda)$ equals the order of the submatrix, it is necessary to compute a second Sturm sequence at $\lambda = \frac{1}{2}(\hat{\lambda}_j + \hat{\lambda}_{j+1})$. If this value is also equal to the order of the submatrix, then $\hat{\lambda}_j$ is not an eigenvalue of the submatrix and should not be used as the shift.

To prevent the use of the shift $\hat{\lambda}_j$ once the eigenpair $(\hat{\lambda}_j, \hat{u}_j)$ has been computed, $\hat{\lambda}_j$ is marked as used when the converged diagonal element $\hat{\alpha}_1$ is closer to $\hat{\lambda}_j$ than to any other computed eigenvalue.

The columns of the accumulated matrix of rotations are the eigenvectors of T corresponding to the eigenvalues in the order that they appear along the diagonal of the converged diagonal matrix. This order is not always the order in which the eigenvalues are used as shifts.

4.4.2 An Experimental Comparison of Some QL Methods

This section offers an experimental comparison of four implementations of the QL algorithm. TQL2 [68] is an implementation of the explicit QL algorithm using Wilkinson's shift. IMTQL2 [68] is the implicit QL algorithm using Wilkinson's shift. The remaining are implementations of the perfect-shift method using Sturm sequence evaluations. PSQL-B uses eigenvalues computed by BISECT in increasing order; PSQL-Q uses eigenvalues from IMTQL1 in the order they are produced before sorting. Both perfect-shift codes are modifications of IMTQL2. All experiments were run on a single Sequent Symmetry S81 processor using the Weitek 1167 floating-point accelerator. The test matrices introduced in Chapter 2 are used in the comparison.

Table 4.1 compares the total times (for eigenvalue and eigenvector computations) and accuracies for PSQL-B, PSQL-Q, TQL2, and IMTQL2 when $n = 100$. For all methods and test problems, the residuals are less than 10^{-14}, and the orthogonalities are less than 10^{-13}. The average number of iterations in TQL2 is measured by $\mathcal{N} = \frac{1}{n}\sum_{i=1}^{m} n_i$, where n_i is the submatrix order at iteration i, and m is the total number of iterations. Because splitting or deflation generally occurs after 1 to 4 iterations regardless of submatrix order, a small value of $\mathcal{N}$ reflects that most of the work is done with small submatrices. A large value of $\mathcal{N}$ corresponds to large submatrices and little splitting.

The differences between TQL2, IMTQL2, and PSQL-Q are explained by the order index and by the amount of time spent in eigenvector calculation. For matrix [1,2,1], TLQ2 and IMTQL2 take roughly equal time; PSQL-B is slower than TQL2, and PSQL-Q is faster. For IMTQL2, $\mathcal{N} = 112$, and for TQL2, $\mathcal{N} = 114$, as compared to $\mathcal{N} = 86$ for PSQL-Q. Thus PSQL-Q is faster because the matrix splits or deflates more than for IMTQL2 or TQL2. The smaller subproblems lead to a faster runtime. As PSQL-B and PSQL-Q take roughly the same time to compute the eigenvectors of this matrix (20.58 seconds and 20.88 seconds, respectively), the time difference for the perfect-shift QL schemes is due to eigenvalue computation. For both PSQL-B and PSQL-Q and all test problems, the cost for Sturm sequence evaluations during eigenvector computation is less than 5% of the total runtime.

For the random matrix, PSQL-B and PSQL-Q both take about one third again as long as TQL2 or IMTQL2. In this case, much more deflation takes place in the eigenvector computation by the perfect-shift methods than by the Wilkinson shift methods. The slowness of both perfect-shift methods can be attributed to the extra cost of eigenvalue computation.

For the modified matrix [1,2,1], the perfect-shift QL method is no longer an efficient alternative to the QL method with Wilkinson's shift. In this case, PSQL-Q and PSQL-B have order indices $\mathcal{N}$ roughly three halves those of TQL2 and IMTQL2 meaning that significantly less splitting occurs with the perfect-shift QL method. As a result, PSQL-Q and PSQL-B take 1.4 and 1.8 times as long as TQL2, respectively.

These data are representative of those obtained for all test matrices (orders 10–512) given in Chapter 2. At worst, PSQL-Q takes about twice the time of TQL2 and, at best, about 0.8 the time. These results do not prove that PSQL-Q

	PSQL with BISECT	PSQL with IMTQL1	TQL2	IMTQL2
Matrix [1,2,1]:				
$\frac{\text{time for method}}{\text{time for TQL2}}$	1.2	0.9	1.0	1.0
$\mathcal{N} = \frac{1}{n}\sum_{i=1}^{m} n_i$	84	86	114	112
$\mathcal{R} = \max_i \parallel T\hat{u}_i - \hat{\lambda}_i\hat{u}_i \parallel_2$	2.26d-15	1.56d-15	2.40d-15	1.61d-15
$\mathcal{O} = \parallel \hat{U}^T\hat{U} - I \parallel_\infty$	4.99d-14	3.77d-14	3.72d-14	3.53d-14
Random matrix:				
$\frac{\text{time for method}}{\text{time for TQL2}}$	1.3	1.3	1.0	1.0
$\mathcal{N} = \frac{1}{n}\sum_{i=1}^{m} n_i$	104	105	123	124
$\mathcal{R} = \max_i \parallel T\hat{u}_i - \hat{\lambda}_i\hat{u}_i \parallel_2$	3.03d-15	3.41d-15	7.41d-15	2.80d-15
$\mathcal{O} = \parallel \hat{U}^T\hat{U} - I \parallel_\infty$	2.44d-14	2.37d-14	8.21d-14	5.92d-14
Modified matrix [1,2,1]:				
$\frac{\text{time for method}}{\text{time for TQL2}}$	1.8	1.4	1.0	1.0
$\mathcal{N} = \frac{1}{n}\sum_{i=1}^{m} n_i$	144	143	97	97
$\mathcal{R} = \max_i \parallel T\hat{u}_i - \hat{\lambda}_i\hat{u}_i \parallel_2$	2.78d-15	5.95d-14	3.33d-15	1.81d-15
$\mathcal{O} = \parallel \hat{U}^T\hat{U} - I \parallel_\infty$	1.14d-14	2.78d-14	3.50d-14	3.62d-14

Table 4.1: Relative times for PSQL, TQL2, and IMTQL2, average number of QL iterations, and accuracy for matrix $[1, 2, 1]$, random matrices, and the modified [1,2,1] matrix of order 100. The order index $\mathcal{N}$ is the scaled sum of matrix orders used at each iteration.

and PSQL-B cannot be faster than TQL2 but only indicate that for a variety of matrices, there is no great advantage to the perfect shift.

Similar results have been observed by the developers of LAPACK for the variant of the implicit perfect-shift QL method that computes each eigenpair in turn [17]. In this version, once an eigenvalue has been computed using the Wilkinson shift, that eigenvalue is immediately used as the shift to compute its eigenvector. When the eigenpair converges, computation continues with the remaining unreduced submatrix. In numerical tests, the time for computing the eigendecomposition of T by this implementation was generally within about 10% (slower or faster) than IMTQL2 [17]. The accuracies were comparable. Thus, for serial computation, no tested implementations of perfect-shift QL provide a clear advantage over TQL2 or IMTQL2.

Chapter 5

Solving the Symmetric Tridiagonal Eigenproblem on the Hypercube

The preceding chapters identify Cuppen's divide and conquer method, bisection with inverse iteration, and the QL method as numerically accurate methods for the symmetric tridiagonal eigenproblem suitable for parallel implementation. The first two methods have been studied on shared-memory multiprocessors and their simulators in [8, 24, 55] and on the grid-based, bit-sliced ICL DAP [4]. A parallel implementation of the QL method for shared-memory multiprocessors has been described in [65]. This chapter concerns solution of the symmetric tridiagonal eigenproblem on a hypercube multiprocessor.

Implementations of Cuppen's method and bisection with inverse iteration for the hypercube are given in Sections 5.1 and 5.2, respectively. Experimental results are also presented in those sections. A comparison of the parallel implementations comprises Section 5.3. The chapter concludes with a discussion of the parallelism of the QL method in Section 5.4. Time complexity analysis shows that the QL method theoretically is not competitive with bisection and inverse iteration on the statically scheduled hypercube.

5.1 Cuppen's Divide and Conquer Method

The recursive nature of Cuppen's method suggests its suitability to implementation on a hypercube. Following the divide and conquer strategy described in Chapter 4, the matrix T is written

$$T \equiv T_{20}$$

$$= \begin{pmatrix} \begin{pmatrix} T_{00} & 0 \\ 0 & T_{01} \end{pmatrix} + \beta_{10} b_{10} b_{10}^T & 0 \\ 0 & \begin{pmatrix} T_{02} & 0 \\ 0 & T_{03} \end{pmatrix} + \beta_{11} b_{11} b_{11}^T \end{pmatrix} + \beta_{20} b_{20} b_{20}^T . (5.1)$$

This subdivision or "tearing" process is repeated and rank-one updating procedures applied recursively. At the ith subdivision, a submatrix T_{ij} is split into

$$T_{ij} = \begin{pmatrix} T_{i-1,2j} & \\ & T_{i-1,2j+1} \end{pmatrix} + \beta_{ij} b_{ij} b_{ij}^T, \; 0 \leq j \leq 2^{d-i} - 1.$$

The number of subdivisions needed to solve the problem is equal to the dimension of the hypercube. At step j, 2^{d-j} j-cubes independently solve eigensystems of order $k2^j$. Upon completion of step d, each processor contains k of the $n = k2^d$ eigenvalues of the original matrix T as well as the k corresponding eigenvectors (of length n).

The assignment of subcubes to processors during this procedure is illustrated for the case $d = 3$ in Table 5.1. The matrix $T \equiv T_{30}$ of order $n = k2^3$ is recursively divided into eight tridiagonal matrices $T_{00}, T_{01}, \ldots, T_{07}$ of order k, and matrix T_{0i} is assigned to processor P_i. In general, the entries T_{ji} are those matrices whose eigensystems are computed in step j by subcube i. The brackets distinguish the subcubes occupied by each eigensystem.

	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7
step 0:	[T_{00}]	[T_{01}]	[T_{02}]	[T_{03}]	[T_{04}]	[T_{05}]	[T_{06}]	[T_{07}]
step 1:	[T_{10}]		[T_{11}]		[T_{12}]		[T_{13}]	
step 2:	[T_{20}]				[T_{21}]			
step 3:	[$T_{30} \equiv T$]							

Table 5.1: Assignment of submatrices to the processors of a 3-cube for the divide and conquer method.

5.1.1 The Algorithm

More precisely, during step 0 processor P_i (a 0-cube) computes the eigensystem (Λ_{0i}, X_{0i}) of the matrix T_{0i}. Because each rank-one updating step requires the eigensystems of two smaller matrices, processors must pair up in step 1 and exchange information within 1-cubes to compute the eigensystems of the four order-$2k$ matrices $T_{10}, \ldots, T_{13}$. After exchanging information about the eigensystems of matrices T_{00} and T_{01}, processors P_0 and P_1 together compute the eigensystem (Λ_{10}, X_{10}) of T_{10}. Processor P_0 holds the leading k eigenvalues and eigenvectors of T_{10}, while P_1 holds the trailing k. The remaining steps proceed in a similar fashion until, at the end of step 3, each of the eight processors contains k eigenvalues and k eigenvectors of length $8k$ of the original matrix $T \equiv T_{30}$. For $0 \leq i \leq 7$, P_i holds eigenvectors indexed $ik+1, \ldots, (i+1)k$.

At the start of Cuppen's method, each node needs a sequence of diagonal and off-diagonal elements of the matrix T. Finding the eigensystem of T_{20} in equation (5.1) on a 2-cube, for instance, requires that the submatrix T_{00} as well as the off-diagonal elements β_{10} and β_{20} be available in processor P_0. Algorithm 5.1.1 details the steps in the determination of all eigenvalues and eigenvectors of a matrix T of order $n = k2^d$ on a d-cube. Only steps 2.a and 2.d, involve data communication.

To begin, the host processor recursively divides the matrix $T \equiv T_{d0}$ d times and allocates submatrix T_{0i} and the d appropriate off-diagonal elements to processor P_i, $0 \leq i \leq 2^d - 1$. Then for j steps, $1 \leq j \leq d$, the cube splits into 2^{d-j} j-cubes which independently compute the eigensystems (Λ_{ji}, X_{ji}) of the matrices T_{ji} of order $k2^j$ using the eigensystems from step $j-1$:

$$\begin{aligned} T_{ji} &= \begin{pmatrix} T_{j-1,2i} & \\ & T_{j-1,2i+1} \end{pmatrix} + \beta_{ji} b_{ji} b_{ji}^T \\ &= \begin{pmatrix} X_{j-1,2i} & \\ & X_{j-1,2i+1} \end{pmatrix} \left[\begin{pmatrix} \Lambda_{j-1,2i} & \\ & \Lambda_{j-1,2i+1} \end{pmatrix} + \rho_{ji} z_{ji} z_{ji}^T \right] \begin{pmatrix} X_{j-1,2i} & \\ & X_{j-1,2i+1} \end{pmatrix}^T . \end{aligned}$$

Denote by S a subcube of dimension j and index i which consists of processors $i2^j$ through $(i+1)2^j - 1$. For simplicity, denote these processors by

$P_0, P_1, \ldots, P_{2^j-1}$ and replace subscripts of the form $(j-1, 2i)$ with 0 and $(j-1, 2i+1)$ with 1. At step j, with the new notation, processors $P_0, P_1, \ldots, P_{2^j-1}$ compute the eigensystem (Λ, X) of

$$T = \begin{pmatrix} T_0 & \\ & T_1 \end{pmatrix} + \beta bb^T = \begin{pmatrix} X_0 & \\ & X_1 \end{pmatrix} \left[\begin{pmatrix} \Lambda_0 & \\ & \Lambda_1 \end{pmatrix} + \rho zz^T \right] \begin{pmatrix} X_0 & \\ & X_1 \end{pmatrix}^T$$

from (Λ_0, X_0) and (Λ_1, X_1). At the beginning of step j, processor P_l for $0 \le l \le 2^{j-1}-1$ contains eigenvalues $lk+1, \ldots, (l+1)k$ of T_0 and columns $lk+1, \ldots, (l+1)k$ of X_0 determined in step $j-1$. These 2^{j-1} processors form a subcube of dimension $j-1$. Similarly, processor P_l for $2^{j-1} \le l \le 2^j - 1$ contains eigenvalues $lk+1, \ldots, (l+1)k$ of T_1 and columns $lk+1, \ldots, (l+1)k$ of X_1. These 2^{j-1} processors also form a subcube of dimension $j-1$.

Algorithm 5.1.1 (Solution of Eigenproblem of Order $n = k2^d$ on a d-Cube.) *In parallel, do on all processors P_i, $0 \le i \le p-1$:*

1. *Compute the eigensystem (Λ_{0i}, X_{0i}) of the matrix T_{0i} of order k using TQL2. (The diagonal of Λ_{0i} contains the eigenvalues of T_{0i} in descending order, and the columns of X_{0i} are the corresponding eigenvectors.)*

2. *For j, $1 \le j \le d$:*

   (a) *Join with $2^j - 1$ other processors to form a subcube of dimension j called S.*

   (b) *Using Algorithm 3.3.1, exchange the elements of Λ_0 and Λ_1 and the elements of the last row of X_0 and the first row of X_1, so that each processor in S contains Λ_0, Λ_1, the last row of X_0, and the first row of X_1.*

   (c) *Compute z and ρ from the last row of X_0, the first row of X_1, and β_{ji}.*

   (d) *Determine a permutation matrix J by merging the sorted sequences diag(Λ_0) and diag(Λ_1) so that the diagonal elements of*

   $$D = J^T \begin{pmatrix} \Lambda_0 & \\ & \Lambda_1 \end{pmatrix} J$$

   are sorted in descending order.

(e) *Permute the elements of* z *accordingly:* $z \Leftarrow J^T z$.

(f) *Apply the product of plane rotations* G *to zero out the elements in* z *that correspond to close elements in* D.

(g) *Identify small elements of* z*, and deflate the problem.*

(h) *Compute elements* $ik+1, \ldots, (i+1)k$ *of* Λ *(eigenvalues of* $D + \rho z z^T$*) by finding the roots of*

$$w(\lambda) = 1 + \rho z^T (D - \lambda)^{-1} z. \tag{5.2}$$

Compute the associated eigenvectors $u_{lk+1}, \ldots, u_{(l+1)k}$ *from*

$$u_j = \frac{(D - \lambda_j)^{-1} z}{\| (D - \lambda_j)^{-1} z \|_2}. \tag{5.3}$$

When $\zeta_m = e_m^T z$ *is small,* $\lambda_m = \delta_m$ *and* $u_m = e_m$.

(i) *Update the eigenvectors:* $(v_{lk+1}, \ldots, v_{(l+1)k}) = G^T (u_{lk+1}, \ldots, u_{(l+1)k})$.

(j) *By means of Algorithm 3.3.2, determine* k *columns of* X *via*

$$(x_{lk+1}, \ldots, x_{(l+1)k}) = \left[\begin{pmatrix} X_0 & \\ & X_1 \end{pmatrix} J \right] (v_{lk+1}, \ldots, v_{(l+1)k}).$$

The deflation rules for finite precision arithmetic developed in [24] described in Chapter 4 were used in this implementation. A hypercube implementation in which the accumulated vectors are stored by rows is described in [14].

5.1.2 Experimental Results

This section presents an experimental evaluation of deflation. Although deflation occurs for most matrices [24], the savings in runtime are reduced in the hypercube implementation. This loss is evident in the speedups $\mathcal{S} = \frac{T_1}{T_{32}}$ measured for two matrices having different amounts of deflation.

Figure 5.1 and Table 5.2 show speedup on 32 processors as a function of matrix order for matrices [1,2,1] and [1,μ,1]. Because of deflation, TREEQL run on one

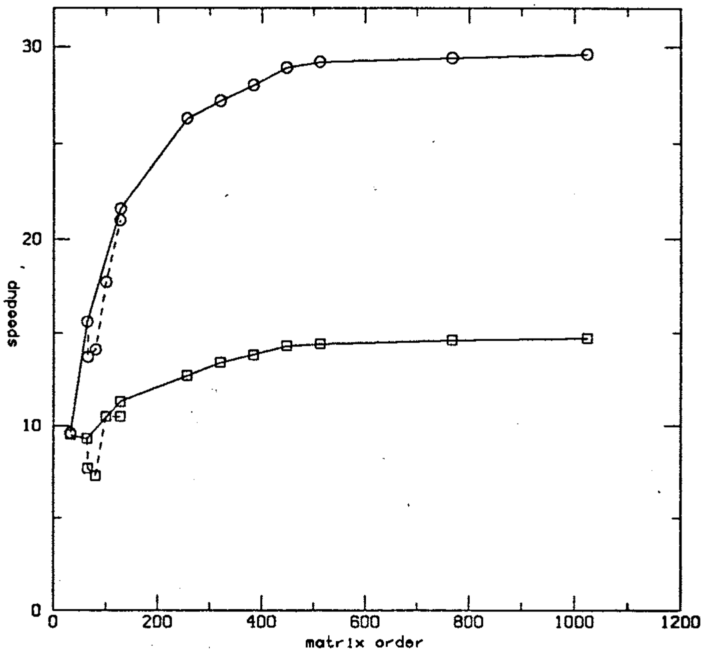

Figure 5.1: Cuppen's method on a 5-cube: speedup $\mathcal{S} = \frac{T_1}{T_{32}}$ for [1,2,1] (squares) and [1,μ,1] (circles) *versus* matrix order. Points for matrix orders that are multiples of 32 are connected with solid lines. Other points are connected with dotted lines.

matrix order	speedup [1,2,1]	speedup [1,μ,1]
32	9.5	9.9
64	9.3	16.6
128	11.3	22.8
256	12.7	26.3
512	14.4	29.2

Table 5.2: Cuppen's method on a 5-cube: speedup $\mathcal{S} = \frac{T_1}{T_{32}}$ for matrices [1,2,1] and [1,μ,1] for several matrix orders.

matrix order	fraction of time in communication
32	.61
64	.36
128	.25
256	.16
512	.16

Table 5.3: Cuppen's method on a 5-cube: fraction of total time spent in communication for matrix [1,2,1] of several orders.

processor is much faster for matrix [1,2,1] than for matrix $[1, \mu, 1]$. TREEQL run on more than one processor, however, takes approximately the same time for both matrices. Thus, although near maximal speedup occurs in the case of little deflation, speedup of only about one half is seen when deflation is prevalent.

The failure to take advantage of deflation is due largely to the static scheduling of processors. The processors no longer solve identical problems at each step when the cube dimension is larger than one. Nevertheless, the data exchange requirements of Algorithm 5.1.1 synchronize the processors. A single processor may encounter significant savings when deflation occurs, but the gain may not be shared by the cube as a whole. Unless the effects of deflation are evenly distributed over the processors of the cube, any time gained during root-finding by a single processor will be lost as it waits for the slower processors during the data exchange routines. In addition, Algorithm 3.3.2 synchronizes processors so that savings in updating the deflated matrix are lost. On a serial machine or on a shared-memory machine with root-finding tasks dynamically scheduled, the savings from deflation can be substantial. In the latter case, the processors operate asynchronously so there is little processor idle time while solving the deflated problem or multiplying matrices to update vectors. On a hypercube, however, a dynamic scheduler incurs additional overhead and is can involve potentially expensive communication of eigenvectors.

The remaining loss of performance is due to communication. At some points in the algorithm, it is more efficient to allow all processors to perform the same

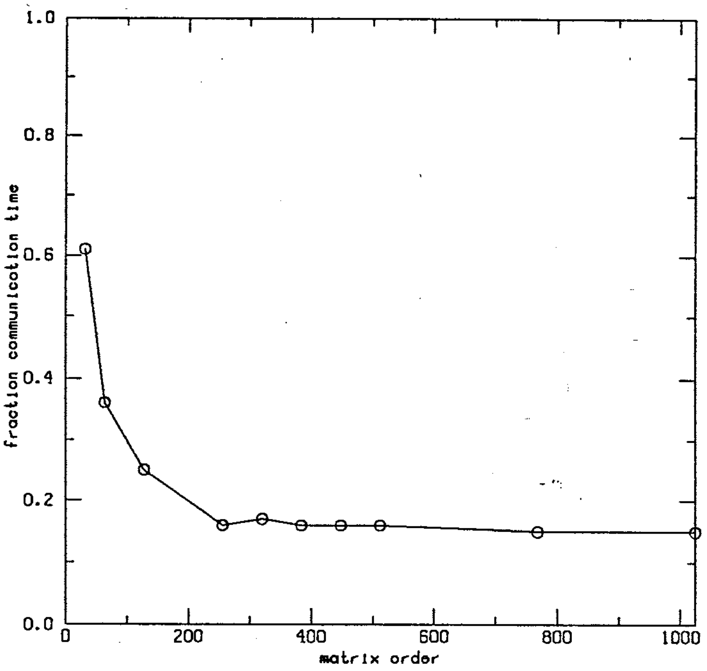

Figure 5.2: Cuppen's method on a 5-cube: fraction of total time spent in communication *versus* matrix order for matrix [1,2,1].

computation than it is to broadcast the results. (The application of rotations in step j.2.(d) of Algorithm 5.1.1 is an example). The communication required for the hypercube similarly reduces the time savings due to deflation. Figure 5.2 and Table 5.3 show the fraction of time spent in communication on a 5-cube for various orders of matrix [1,2,1]. Computation time is shown to be greater than communication time for matrix orders as small as 64. For eight or more eigenvectors per processor, the communication cost levels off at about 16% of the total time.

5.2 Bisection with Inverse Iteration

5.2.1 The Algorithm

The hypercube implementation of bisection involves a straightforward partitioning of the computing tasks outlined in Algorithm 5.2.1. For a matrix of order $n = kp$, each processor computes k eigenvalues and eigenvectors. Eigenvalues are computed using the EISPACK routine TRIDIB. This implementation of bisection allows computation of any number of consecutive eigenvalues specified by their indices [68]. Processor P_i uses TRIDIB to compute eigenvalues $ik + 1$ through $(i + 1)k$.

When all eigenvalues are well-separated, each eigenvector can be computed with a small number of inverse iterations. An effective load balance is then achieved by computing $\frac{n}{p}$ of the vectors in each processor. No communication is required, and an equal number of eigenvectors is stored in each processor. When close eigenvalues occur, the rate of convergence of inverse iteration decreases, and the corresponding eigenvectors must be orthogonalized.

The $p = 2^d$ hypercube processors are numbered in a ring according to a Gray code ordering. Each processor examines the complete list of eigenvalues and perturbs those too close for inverse iteration. Processor j employs the EISPACK routine TINVIT [68] to compute eigenvectors corresponding to (possibly perturbed) eigenvalues indexed $j + 1, j + p + 1, \ldots, j + kp + 1 \leq n$. The eigenvectors corresponding to close eigenvalues are reorthogonalized using Algorithm **3.3.3** for the Modified Gram-Schmidt procedure.

Bisection with inverse iteration on the hypercube is given as Algorithm 5.2.1. To begin, each processor has all diagonal and off-diagonal elements of the matrix.

Algorithm 5.2.1 (Solution of Eigenproblem of Order $n = kp$ on a p-Processor Hypercube.)

In parallel, do on all processors P_i, $0 \leq i \leq p-1$:

1. *Determine initial search area:*

   Compute all Gerschgorin disks to find the interval containing all n eigenvalues.

2. *Compute eigenvalues:*

   Use bisection to determine the $\frac{n}{p}$ eigenvalues $ik+1$ through $(i+1)k$.

3. *Communicate eigenvalues:*

   Exchange computed eigenvalues with all other processors using the alternate direction exchange of Algorithm 3.3.1.

4. *Perturb eigenvalues:*

   Sort the n eigenvalues and perturb any spaced too closely for inverse iteration.

5. *Compute eigenvectors:*

   Compute the $\frac{n}{p}$ eigenvectors corresponding to eigenvalues indexed $i+1, i+p+1, \ldots, i+kp+1 \leq n$. Employ Algorithm 3.3.3 to orthogonalize eigenvectors corresponding to close eigenvalues.

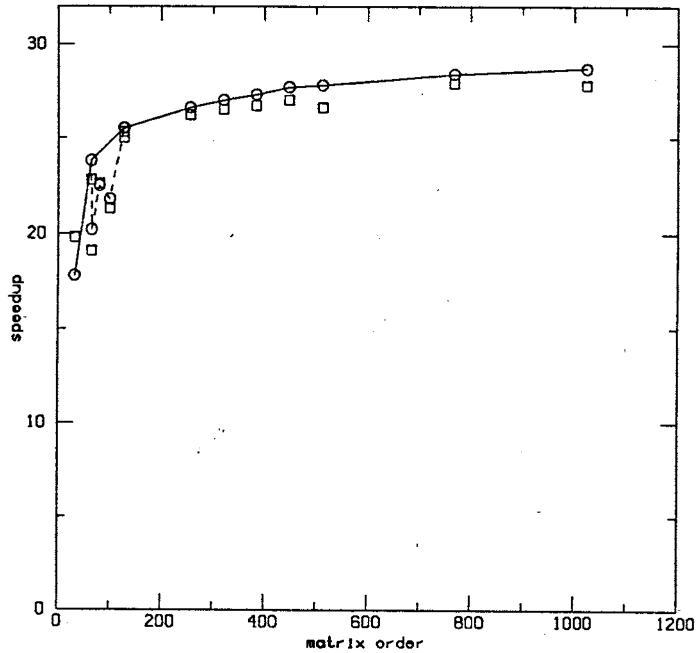

Figure 5.3: Bisection and inverse iteration on an iPSC/d5M: speedup for [1,2,1] (circles) and random matrices (squares) *versus* matrix order.

5.2.2 Analytical and Experimental Results

As noted in the preceding sections, both bisection and inverse iteration are readily implemented on local-memory multiprocessors. The efficiency of this approach is reflected in the plots given in Figure 5.3. (The same data is also presented in Table 5.4.) The speedup equals the time for EISPACK's TRIDIB with TINVIT run on a single processor divided by the greatest node time for the hypercube bisection and inverse iteration procedures executed on a 32-node iPSC[1]. Different random matrices were generated for each order, so no relation is expected between random matrix data points.

The speedup for [1,2,1] increases smoothly for matrix orders proportional to the number of processors. Efficiencies ranging from 77% to 89% are achieved for

[1] TRIDIB with TINVIT is the fastest method for finding all eigenvalues and eigenvectors of matrix [1,2,1] and of the random test matrices of orders at least 32 on one processor of the iPSC/1-d5M. For order 512 and matrix [1,2,1], TRIDIB and TINVIT take a total of 2340.0 seconds.

matrix order	speedup [1,2,1]	speedup random matrix
32	18.7	19.8
64	23.8	22.8
128	25.5	25.0
256	26.6	26.2
512	27.8	26.6

Table 5.4: Bisection and inverse iteration on a 5-cube: speedup $\mathcal{S} = \frac{T_1}{T_{32}}$ for matrix [1,2,1] and a random matrix for several orders.

matrix orders above 100. Comparable speedups for random matrices of all orders suggest that the results are not strongly dependent on properties particular to matrix [1,2,1].

Efficiencies for other matrix orders fall to as much as 12% below the smooth line of those for multiples of 32 because some processors are required to compute one more eigenvalue and eigenvector than the others. The alternate direction exchange of eigenvalues (in step 3 of Algorithm 5.2.1) synchronizes the processors. Those processors with a smaller workload are idle until the processors with a greater workload enter the exchange. The orthogonalization of eigenvectors can be similarly delayed by the uneven distribution of inverse iteration tasks. Hence, the time to complete the parallel computation is determined by the processor with the largest assignment of work.

Despite the parallelism inherent in the bisection and inverse iteration procedures, maximum speedup is not achieved because of non-arithmetic tasks and non-parallel computation. The contribution of this overhead is comparable in magnitude to the reduction in speedup. Figure 5.4 and Table 5.5 show the average fraction of the total time spent idle or in communication by one processor. This time was determined by summing all time spent in arithmetic and subtracting it from the total time. Again, the points for matrix [1,2,1] measured at orders divisible by 32 define a smooth curve falling from 20% of the total time at matrix order 32 to about 2% of the total for matrix orders larger than 320.

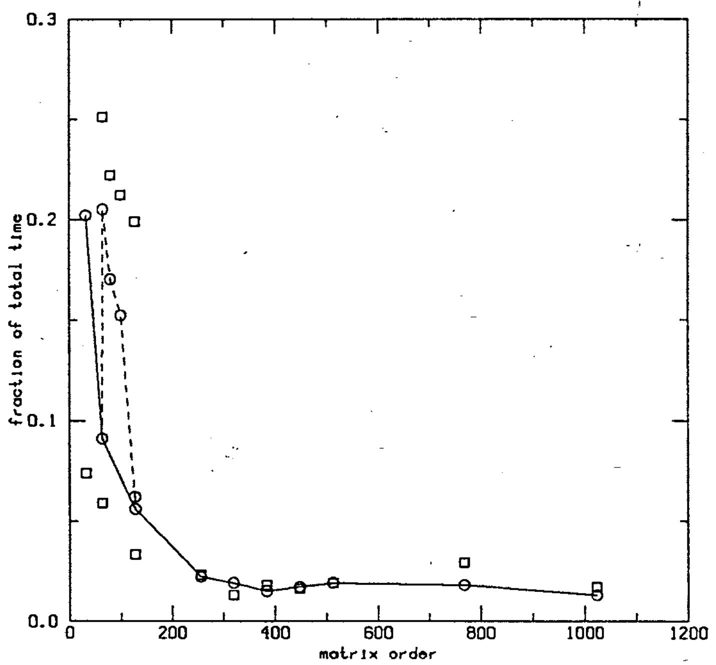

Figure 5.4: Bisection on an iPSC/d5M: communication overhead as a fraction of the total time *versus* Matrix Order.

The increase in processor idle time when n is not divisible by 32 accounts for the larger fractions for these orders. As expected, this increase is significant for order 65 but barely discernible for order 127.

The eigenvalue computation begins with each processor independently computing all Gerschgorin disks and then carrying out an initial bisection to determine the interval containing its share of the eigenvalues. While this process could be implemented in parallel, its present contribution to the total time is small. For a variety of matrices including matrix [1,2,1], the initial bisection time decreases smoothly for all matrix orders from a maximum of about 19% of the total computation time at order 32 to about 4% at order 512 and 1% at order 1024. The grouping of close eigenvalues done in each processor occupies less than 5% of computation time for all matrix orders. The idle time in Algorithm 3.3.3 (MGS) represents an additional loss of efficiency amounting to less than 2% total time for all orders of matrix [1,2,1]. Thus, at order 32, about 42% of the total

matrix order	fraction of time in communication [1,2,1]	fraction of time in communication random matrix
32	.074	.202
64	.059	.091
128	.033	.056
256	.023	.022
512	.019	.019

Table 5.5: Bisection on an iPSC/d5M: communication overhead as a fraction of the total time for several matrix orders.

time is spent in communication and non-parallel arithmetic. This slowing of the parallel execution time completely accounts for the observed efficiency of 58%. Similarly, the 13% of the total time spent in overhead for order 512 approximates the observed lowering of the speedup curve.

Figure 5.5 shows the average fraction of the total time spent in computing, exchanging, and grouping eigenvalues, in determining eigenvectors, and in orthogonalizing eigenvectors corresponding to closely spaced eigenvalues, for several orders of matrix [1,2,1]. Finding and distributing the eigenvalues to all processors occupies more than 80% of the total time, while computing the eigenvectors occupies most of the remaining time.

5.2.3 A Model Problem

While the exact time distribution among computing tasks is problem dependent, analysis of a simple model problem sheds light on the expected arithmetic requirements. Consider the symmetric, tridiagonal matrix T_{model} of order $n = kp$ having eigenvalues $0, \frac{\alpha}{n}, \ldots, \frac{(n-1)\alpha}{n}$. The spacing of the eigenvalues is assumed wide enough that additional orthogonalization of eigenvectors is not required. Suppose that its n Gerschgorin disks overlap to form a continuous interval from 0 to α. In this way, the spectrum of T_{model} approximates that of matrix [1,2,1] which has n eigenvalues initially confined within an interval of length four for all values of n.

During the initial bisection of Algorithm 5.2.1, each processor finds an interval of length $\frac{\alpha}{p}$ containing k eigenvalues. This step takes takes $l \approx \log_2 p$ iterations for a total of $l+1$ Sturm sequence evaluations. Computation of the k eigenvalues takes $\sum_{j=1}^{k} \log \frac{\alpha j}{n\delta} + 1$ Sturm sequence evaluations. In subsequent steps, each processor extracts its k eigenvalues. If the time to complete one Sturm sequence evaluation is $2n\omega$, then the total time for the eigenvalue computation is

$$\begin{aligned} \tau_B &\approx \left[(l+1) + \sum_{i=1}^{k} \left(log_2 \frac{\alpha i}{n\delta} + 1\right)\right] 2n\omega \\ &\approx \left(\log_2 p + 1 + k \log_2 \frac{\alpha k}{n\delta} + k\right) 2n\omega. \end{aligned}$$

The time to perform two inverse iterations for each of k eigenvectors within each processor is $\tau_I = 10nk\omega$.

The attainable tolerance δ is related to both the machine precision and the eigenvalue of largest magnitude [68]. For the model problem in double precision, $\delta \approx 10^{-15}\alpha \approx 2^{-50}\alpha$. Thus, for orders 32 through 1024 on a 5-cube, $\frac{\tau_B}{\tau_I}$ decreases from 12 to about 11 indicating that the eigenvalue computation dominates the total arithmetic time. The eigenvalues of matrix [1,2,1] are more closely spaced than those of T_{model} and so require fewer bisections to extract. Thus, the eigenvalue computation for [1,2,1] takes only about about four times as long as the eigenvector computation. (As indicated by Figure 5.4, non-arithmetic operations contribute minimally to the experimental result.)

Figure 5.5 shows that the ratio of eigenvalue to eigenvector computation times decreases for matrix [1,2,1] as it does for T_{model}. For [1,2,1], $\frac{\tau_B}{\tau_I}$ falls from 7.1 at order 32 to 4.3 at order 1024.

The ratios of communication and arithmetic times for both T_{model} and matrix [1,2,1] are small and decrease with matrix order. Under the assumption that $\beta \approx 10\omega$ and $\tau \approx \frac{10\omega}{125}$, the communication time is the time for an alternate direction exchange

$$\begin{aligned} \tau_C &= 2\left[\log_2 p\beta + (p-1)\frac{n}{p}\tau\right] \\ &\approx 2\left(10\log_2 p + \frac{10n}{125}\right)\omega. \end{aligned}$$

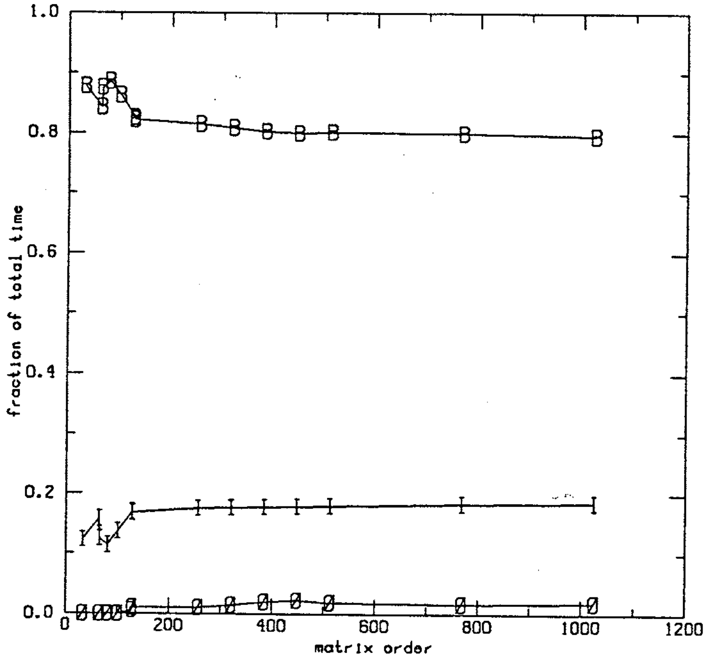

Figure 5.5: Bisection on a 5-cube: Fraction of total time spent in eigenvalue computation (B), eigenvector computation (I), and orthogonalization (O) *versus* matrix order.

For the model problem on a 5-cube, the ratio of communication time to computation time $\frac{\tau_C}{\tau_B}$ falls from .02 at order 32 to 3×10^{-5} at 1024. The fraction of communication time shown for [1,2,1] in Figure 5.4 is greater than the predicted value for T_{model} because the eigenvalue computation takes longer for T_{model}. As shown in Figure 5.5, orthogonalization does not greatly alter the run time for matrix [1,2,1].

5.2.4 Distribution of eigenvectors

The cyclic distribution of eigenvectors was chosen for its simplicity and systematic arrangement of eigenvectors across processors. Unless a cluster is very large (including more than p eigenvalues), no processor is required to handle more than one vector from any given cluster. Furthermore, the burden of cluster handling is generally spread across processors.

The main drawback of the cyclic distribution is that processors are synchronized by the alternate direction exchange of eigenvalues. Thus, all processors must wait until the last processor has computed its eigenvalues before beginning eigenvector computation. Although this idle time did not significantly degrade parallel efficiency for any of the matrices tested, severe load imbalance could result if some processors were assigned only the fast computation of clusters of eigenvalues while others were required to compute well-separated ones. The load imbalance is exacerbated by an uneven distribution of orthogonalization tasks. This section examines two alternative static load balancing schemes.

A weighted task scheduling scheme also involves synchronization of processors and the resultant potential for load imbalance. In this approach, each computational task is assigned a weight or *time value* based on its expected completion time. The weighted tasks are apportioned among processors according to a scheduling rule designed to give a fast completion time [60]. As noted earlier, cluster size can be used as a weighting for approximate load balancing. The eigenvectors associated with a cluster take more iterations and more orthogonalization than the same number of eigenvectors corresponding to well-separated eigenvalues.

The weighted task approach can suffer additional inefficiencies after the synchronization. First, unlike a static allotment of eigenvectors (such as the block distribution), creation and manipulation of the task queue may introduce some computational overhead. Second, if assignment of tasks to processors is based solely on time values, a processor given a cluster by the schedule can be required to compute many fewer vectors than one assigned only eigenvectors corresponding to single eigenvalues. While such assignment can give a balance of time requirements, it gives a poor distribution of memory use. A weighting scheme that takes clustering into account can prevent a storage imbalance, but it leaves the eigenvectors arranged irregularly across processors. The regularity of the cyclic (or the following block) distribution scheme is lost, and a rearrangement of the eigenvectors may be required.

A final scheduling approach maintains the block distribution used for eigenvalue computation, thereby largely avoiding the synchronization of processors. A processor can start most eigenvector computations without having to wait for all other processors to finish their eigenvalue computations [26]. When eigenvalues are close, however, adjacent processors may need to communicate eigenvalues for the perturbation (step 4) in Algorithm 5.2.1, thereby causing processors to be synchronized.

To see when the block strategy is advantageous, consider a model eigenproblem of order $n = 4k$ with $k < 10$ solved on a four-processor hypercube multiprocessor. Suppose that processor P_0 must compute k computationally coincident eigenvalues in an interval of length $\alpha = 1$ while processors $P_1, \ldots, P_3$ each compute k equally spaced eigenvalues in intervals of length α.

P_0 uses bisection to reduce the interval of width α to one of a width $\delta = 10^{-15} \approx 2^{-50}$. This process requires $\lceil \log_2 \frac{\alpha}{\delta} \rceil$ Sturm sequence evaluations for a total time of

$$\tau_0 \approx \left(\log \frac{\alpha}{\delta} \right) 2n\omega.$$

At the same time, processors $P_1, \ldots, P_3$ compute their eigenvalues. Finding its largest eigenvalue first, each processor reduces its interval of width α to one of width δ in $l_1 \approx \log_2 \frac{\alpha}{\delta}$ bisections. After this computation, the portion of the interval following the first eigenvalue computed is discarded, leaving a new search interval of length $\alpha - \frac{\alpha}{k}$. The second eigenvalue is then extracted in $l_2 \approx \log_2 \frac{\alpha - \frac{\alpha}{k}}{\delta}$ bisections. In general, the jth eigenvalue is found in an interval of width $\alpha - (j-1)\frac{\alpha}{k}$ in $l_j \approx \log_2 \frac{\alpha-(j-1)\frac{\alpha}{k}}{\delta}$ bisections, a process requiring $l_j + 1$ Sturm sequence evaluations. The total time for this operation is

$$\begin{aligned} \tau_1 &\approx \sum_{j=1}^{k} (\log_2 \frac{\alpha j}{\delta k} + 1) 2n\omega \\ &\approx k(\log_2 \frac{\alpha k}{\delta k} + 1) 2n\omega \\ &\approx k \log_2 \frac{\alpha}{\delta} 2n\omega \\ &= k\tau_0. \end{aligned}$$

If the eigenvalues are sufficiently distant from the endpoints of the original search intervals, no cluster is spread across processors, and no communication is necessary before the eigenvalue perturbation step.

If $\frac{\alpha}{k}$ is large enough that no reorthogonalization of eigenvectors is necessary for equally-spaced eigenvalues, then the eigenvector computation by processors $P_1, \ldots, P_3$ takes time

$$\tau_1' \approx 10kn\omega,$$

where each of the two inverse iterations takes time $5n\omega$. For the clustered eigenvalues of P_0, three iterations might be needed for each eigenvector, and reorthogonalization is required for all k eigenvectors. The time for eigenvector computation by processor P_0 is then

$$\tau_0' \approx 15kn\omega + 2k^2 n\omega.$$

P_0 finishes its computation of k clustered eigenvalues and their eigenvectors in time

$$\tau_0 + \tau_0' \approx [2log_2 \frac{\alpha}{\delta} + 15k + 2k^2] n\omega.$$

$P_1, \ldots, P_3$ finish in time

$$\tau + \tau_1' \approx [2klog_2\frac{\alpha}{\delta} + 10k]n\omega.$$

When $\frac{\alpha}{\delta} \approx 2^{50}$, P_0 is the fastest processor when $1 < k < 10$. The time for cyclic distribution is the time for the block distribution plus the time for the alternate direction exchange and the modified Gram-Schmidt reorthogonalization in Algorithm 5.2.1.

Suppose instead that the eigenvalues are distributed so that P_0 and P_1 each have $k/2$ evenly spaced eigenvalues in an interval of width $\frac{\alpha}{2}$ and $k/2$ coincident eigenvalues in the remaining half. The k coincident eigenvalues form a single cluster split equally between the two processors. P_2 and P_3 each have k evenly spaced eigenvalues in intervals of width α.

P_2 and P_3 compute their eigenpairs in time $\tau_1 + \tau_1'$. P_0 and P_1 both require time

$$\tau_2 \approx [2\log_2\frac{\alpha}{2\delta} + 15\frac{k}{2} + \frac{k^2}{2}]n\omega$$

to compute the eigenpairs of the cluster and time

$$\tau_2' \approx [k\log_2\frac{\alpha}{2\delta} + 5k]n\omega$$

for the separated eigenpairs. In addition, communication time of

$$\tau_C \approx \frac{k}{2}[\beta + n\tau]$$

is needed to pass $\frac{k}{2}$ orthogonalized eigenvectors from P_0 to P_1 during the modified Gram-Schmidt procedure. On the iPSC/1-d5M, $\beta \approx 10\omega$ and $\tau \approx \frac{10}{125}\omega$, so that

$$\tau_C \approx k(5 + \frac{n}{25})\omega.$$

The time for P_0 and P_1 to compute all eigenpairs without reorthogonalization is

$$\tau_E \approx [(2 + k)\log_2\frac{\alpha}{2\delta} + 12.5k]n\omega.$$

Method	Order	Time (seconds)	Residual $\|TX - \Lambda X\|$	Orthogonality $\|X^TX - I\|$
Cuppen's Bisection	32	1.3 0.6	9.2e-16 9.4e-15	7.0e-16 3.3e-14
Cuppen's Bisection	100	10.5 4.5	1.9e-15 3.3e-14	1.9e-15 3.1e-14
Cuppen's Bisection	512	611.8 88.7	8.4e-15 8.8e-13	1.8e-14 6.0e-13

Table 5.6: Comparison of methods for matrix [1,2,1] on a 5-Cube.

For $\frac{\alpha}{\delta} \approx 2^{50}$, $\tau_C + \tau_2' < \tau_E$ for all values of k. Therefore, P_0 can compute the portion of the eigensystem corresponding to its clustered eigenvectors and transmit the reorthogonalized eigenvectors to P_1 during the time that P_1 performs all computing tasks besides reorthogonalization. The reorthogonalized eigenvectors from P_0 will have arrived once P_1 is ready to begin reorthogonalization of the remaining $\frac{k}{2}$ eigenvectors. Because there is no idle waiting time, P_0 and P_1 complete their tasks in time slightly less than

$$\tau_2 + \tau_2' + \tau_C.$$

For $\frac{\alpha}{\delta} \approx 2^{50}$,

$$\tau_2 + \tau_2' + \tau_C < \tau_1 + \tau_1'$$

whenever $1 < k < 10$. Therefore, the total time for computation is that required by processors P_2 and P_3 and is again faster for the block than for the cyclic computation. If a cluster is spread over more than two processors, a similar sort of load balance can be achieved at the expense of added communication: messages must travel in both directions. Block distribution may thus represent an improvement to the cyclic distribution when clustered eigenvalues occur.

5.3 Comparison

Tables 5.6 and 5.7 show the total time, the residual, and the deviation from orthogonality for several orders of matrix [1,2,1] and of random matrices for the

Method	Order	Time (seconds)	Residual $\|TX - \Lambda X\|$	Orthogonality $\|X^TX - I\|$
Cuppen's Bisection	32	1.1 0.5	2.7e-15 2.9e-15	2.7e-15 3.0e-14
Cuppen's Bisection	100	10.4 4.6	7.4e-15 3.6e-14	8.9e-15 6.5e-14
Cuppen's Bisection	512	623.9 88.3	7.9e-15 5.3e-13	1.3e-14 2.1e-13

Table 5.7: Comparison of methods for random matrices on a 5-Cube.

divide and conquer method and for bisection with inverse iteration. The divide and conquer method gives more accurate results, consistently yielding smaller residuals and orthogonalities than bisection with inverse iteration.

Bisection is the fastest method for finding all the eigenvalues and eigenvectors at all orders. The speedups over the fastest sequential method are problem dependent, but the figures suggest that maximal speedup cannot be expected for either of the methods. Speedup of the divide and conquer method is especially small when significant deflation occurs in some but not all subproblems.

5.4 The QL Method

The purpose of this section is to describe a hypercube implementation of the QL method and to estimate its parallel speedup using complexity analysis. This result is used to show that the QL method is theoretically slower than bisection with inverse iteration on the Intel hypercube. The analysis assumes a symmetric tridiagonal $n \times n$ matrix T and the shifted QL iteration for $k = 0, 1, \ldots$ described in Chapter 4:

$$\begin{aligned} T_0 &= T \\ T_k - \mu_k I &= Q_k L_k \qquad (k \geq 0) \\ T_{k+1} &= L_k Q_k + \mu_k I \\ &= Q_k^T T_k Q_k = \Pi_k^T T_0 \Pi_k. \end{aligned}$$

When perfect shifts are used, they can be computed efficiently in parallel by bisection, for example. When the Wilkinson shift is used, computation of the shift and application of the rotations cannot be overlapped. This suggests that the critical issue in a hypercube implementation is efficient implementation of the $O(n^3)$ application of rotations for eigenvector computation. One straightforward scheme is to use one processor P_0 to carry out the eigenvalue computation by the QL method without accumulating rotations and the remaining processors to accumulate the rotations into the eigenvector matrix. After each iteration, P_0 broadcasts the rotations to the remaining processors.

The time complexity of iteration k is determined as follows. If the iterate after deflation $\bar{T}_{k-1}$ is of order m, P_0 requires time [33]

$$\tau_0 = 14(m-1)\omega$$

to compute and apply the 2×2 rotations $G_1, \ldots, G_{m-1}$ to produce

$$\bar{T}_k = G_{m-1}^T \ldots G_1^T \bar{T}_{k-1} G_1 \ldots G_{m-1} = \bar{Q}_{k-1}^T \bar{T}_{k-1} \bar{Q}_{k-1},$$

where $\bar{T}_{k-1}$ is a diagonal block of T_{k-1} and $T_{k-1} = \Pi_{k-1}^T T_0 \Pi_{k-1}$. If the rotations $G_1, \ldots, G_{m-1}$ are stored as a vector of length $2(m-1)$, they can be transferred from P_0 to all other processors by, for example, sending them around the ring of p processors in time $\tau_1 = (p-1)\beta + 2(m-1)\tau$. Assume that the matrix of rotations Π_{k-1} is stored by rows in the $p-1$ remaining processors. Each processor can then apply the rotations $G_1, \ldots, G_{m-1}$ in turn to its $\frac{n}{p-1}$ rows of Π_{k-1} in time [33]

$$\tau_2(m) = 4(m-1)\frac{n}{p-1}.$$

The effect of this operation is to overwrite Π_{k-1} with

$$\Pi_k = \Pi_{k-1} \begin{pmatrix} I & \\ & \bar{Q}_{k-1} \end{pmatrix}.$$

On the iPSC/1-d5M (32 processors),

$$\tau_0 + \tau_1 \approx (320 + 15m)\omega, \quad \tau_2(m) \approx \frac{mn}{8}\omega.$$

Let $n = 512$. In this case, $\frac{\tau_0+\tau_1}{\tau_2(m)} < 1$ for all $m \geq 2$. Furthermore, the computation of rotations for the deflated submatrix (of order $m-1$ or less) takes less time than the application of rotations to the order m submatrix, and the QL method can be pipelined. P_0 computes and broadcasts the first set of rotations. P_0 then computes and broadcasts the second set of rotations while $P_1, \ldots, P_{p-1}$ accumulate the first set. In general, P_0 can compute the rotations at iteration $k+1$ while the remaining processors accumulate the ones from earlier iterations. The total time for the parallel QL method is then the time for P_0 to start the pipeline plus the time for accumulation of rotations or about

$$\tau_0 + \tau_1 + \sum_{i=1}^{l} \tau_2(m_l), \tag{5.4}$$

where l is the total number of QL iterations, and m_l is the order of the deflated matrix at iteration l. This figure assumes that the time spent receiving messages by processors $P_1, \ldots, P_{p-1}$ is negligible compared to the time to accumulate rotations. Because the total time in equation (5.4) is dominated by the third term, the approximate maximum speedup of the QL method on the hypercube is $p-1$. The speedup can be improved by assigning accumulation tasks to P_0 once it has finished computing all rotations.

This pipeline only works consistently when n is very large. When $n = 100$, for example, $\tau_0 + \tau_1 > \tau_2(m)$ for all $m \leq 100$ and the computation of rotations for iteration k is not overlapped by the application of rotations for iteration $k-1$. Even if n is large enough to allow pipelining, matrix splitting may lower the speedup. That is, if the matrix splits into a small submatrix T_1 and a large submatrix T_2, P_0's computation of rotations for the first iteration with T_2 may take longer than the application of rotations for T_1. In this case, processors $P_1, \ldots, P_{p-1}$ are idle while P_0 computes and broadcasts the rotations. The pipe must be restarted for T_2.

In the best case, however, the parallel QL method has speedup near the number of processors. A different implementation, thus, cannot improve the speedup. The expected parallel time for the implicit shift QL method can then be determined by comparing it with bisection and inverse iteration which has similar maximal speedup. As noted earlier, BISECT with TINVIT is faster, though less accurate, than TQL2 for all tested problems on a single hypercube processor. The serial experiments presented in Chapter 6 confirm that the QL method is also consistently and substantially slower than a higher accuracy implementation of bisection and inverse iteration for a variety of problems. Because bisection with inverse iteration and the QL method both have speedups near the number of processors, the slowness of the latter on one processor indicates that it would not be competitive on more than one processor. Bisection with inverse iteration thus remains the fastest, most parallel method on the hypercube.

Chapter 6

Improving the Accuracy of Inverse Iteration

Chapters 4 and 5 examine methods for accurate solution of the symmetric tridiagonal eigenproblem serially and on a statically-scheduled, distributed-memory multiprocessor. Experimental results presented in those chapters confirm that bisection with inverse iteration is usually the fastest and most efficient parallel eigensolver as well as the fastest serial method for solving large order eigenproblems. As implemented in EISPACK's TINVIT [68], however, inverse iteration leads to less accurate computed eigenvectors than do existing implementations of the QL method [68] or the divide and conquer method [24]. The factors influencing the accuracy of inverse iteration are examined in this chapter.

A basic implementation of inverse iteration for computing the eigenvectors $\hat{U} = (\hat{u}_1, \ldots, \hat{u}_n)$ of the unreduced symmetric tridiagonal matrix $T = U\Lambda U^T$ from the computed eigenvalues $\hat{\lambda}_1, \ldots, \hat{\lambda}_n$ may be outlined as follows.

Algorithm 6.0.1 (Basic Inverse Iteration)

For $j = 1, \ldots, n$:

1. *Choose a starting vector y with $\| y \|_2 = 1$.*
2. *Solve the tridiagonal system $(T - \hat{\lambda}_j)z = y$.*
3. *If the reorthogonalization criterion is satisfied, orthogonalize the iterate z with respect to those previously computed eigenvectors corresponding to computed eigenvalues close to $\hat{\lambda}_j$.*

4. If the stopping criterion is not satisfied, set $y = z$ and go to Step 2.

5. Accept $\frac{z}{\|z\|_2}$ as the computed eigenvector $\hat{u}_j$.

Assuming that Steps 2 and 3 are carried out using stable, accurate methods, the overall accuracy of this algorithm is determined by the choice of starting vector and the criteria for reorthogonalizing vectors or ending iteration. None of these items, however, can be firmly established on theoretical grounds. They are examined experimentally in Section 6.1, and heuristics are selected for each. An implementation of inverse iteration (III) encompassing these improvements is presented in Section 6.2. Section 6.3 compares the accuracy and serial runtimes of TREEQL [24], TQL2 [68], and the combination of BISECT [68] with III.

6.1 Experimental Results

This section focuses on improving the accuracy of inverse iteration. The goal is to experimentally identify changes to the EISPACK routine TINVIT [68] that would allow it to compute eigenvectors as accurately as those produced by the QL routine TQL2 [68] or the divide and conquer routine TREEQL [24]. The effects of the suggested changes on the efficiency of the method are addressed in Section 6.2.

The implementation of TINVIT is described as Algorithm 6.1.1. For a matrix T with diagonal elements $\alpha_1, \ldots, \alpha_n$ and off-diagonal elements $\beta_2, \ldots, \beta_n$, The norm used in the reorthogonalization criterion is defined by

$$\| T \|_R \equiv \max_{j=1,\ldots,n} (|\alpha_j| + |\beta_j|), \ \beta_1 \equiv 0.$$

Note that $\| T \|_R < \| T \|_\infty$ for an unreduced matrix T. Numerical details of TINVIT such as vector scaling to prevent overflow are not presented in Algorithm 6.1.1.

Algorithm 6.1.1 (Outline of TINVIT)

```
For $j = 1, \ldots, n$
```

```
1. Initialize the set of eigenvalues close to $\hat{\lambda}_j$: $CLUSTER(\hat{\lambda}_j) = \emptyset$.
   If $j > 1$ and $|\hat{\lambda}_j - \hat{\lambda}_{j-1}| < 10^{-3} \| T \|_R$, then
       $CLUSTER(\hat{\lambda}_j) = CLUSTER(\hat{\lambda}_{j-1}) \bigcup \{\hat{\lambda}_{j-1}\}$
       $= \{\hat{\lambda}_i, \ldots, \hat{\lambda}_{j-1}\}, i \leq j-1.$
2. Initialize the iterate norm $\sigma \equiv 0$.
3. Loop until the iterate norm $\sigma \geq 1.0$. (Error exit after 5 iterations.)
   3.a Factor $(T - \hat{\lambda}_j) = LU$.
   3.b If this is the first iteration, solve triangular system $Lz_j = e$, where $e$
       is the vector of all ones.
       Otherwise, solve tridiagonal system $(T - \hat{\lambda}_j)z_j = y_j$.
   3.b Reorthogonalize $z_j$ with respect to $\hat{u}_i, \ldots, \hat{u}_{j-1}$.
   3.c Set $\sigma = \| z_j \|_\infty$.
4. Repeat steps 3.a and 3.b once.
5. Accept $\frac{z_j}{\|z_j\|_2}$ as computed eigenvector $\hat{u}_j$.
```

The experiments presented in this section use a version of TINVIT modified to use different starting vectors for each eigenvalue and to perform a specified fixed number of iterations for all computed eigenvectors regardless of whether or not all eigenvectors had converged. A maximum of five iterations was performed in each experiment. All computations are performed in double precision on a single Sequent processor using the Weitek 1167 floating-point accelerator. The experimental results in Chapter 4 show that residuals $\mathcal{R} = \max_i \| T\hat{u}_i - \hat{\lambda}_i \hat{u}_i \|_2$ less than 10^{-14} can generally be achieved for problem orders up to 525 using TQL2 or TREEQL on this processor. Orthogonalities $\mathcal{O} = \| \hat{U}^T \hat{U} - I \|_\infty$ can have values less than 10^{-14} for orders near 32, less than 10^{-13} for orders near 100, and less than 10^{-12} for orders near 512.

Matrix [1,2,1] of orders 32, 100, and 512 and the glued Wilkinson matrix W_g^+ of orders 42, 105, and 525 are used to illustrate the accuracy effects. Matrix [1,2,1] has computationally distinct eigenvalues for all orders tested, and W_g^+ has strongly clustered ones. Conclusions drawn from these two examples are confirmed using collections of random matrices with uniformly distributed diagonal and off-diagonal elements between -1 and 1 and of matrices formed by applying orthogonal transformations to some diagonal matrices with clustered diagonal elements. The random matrices have well-separated eigenvalues. All test matrices were introduced in Chapter 2.

6.1.1 Starting Vectors

Using the orthogonal eigenvectors of the matrix T as basis vectors, the starting vector y may be written $y = \sum_{j=1}^{n} \eta_j u_j$. Suppose $\lambda = \hat{\lambda}_k$ is a computationally distinct eigenvalue, then a good starting vector for computing $\hat{u}_k$ is one with a significant component in the u_k direction. A good starting vector thus has a large value of $|\eta_k|$ relative to the other components. The largest component of the iterate $z = \sum_{j=1}^{n} \frac{\eta_j}{\lambda - \hat{\lambda}_j} u_j$ derived from this starting vector is then in the direction of u_k. When $\lambda = \hat{\lambda}_k = \ldots = \hat{\lambda}_m$, a good starting vector requires large coefficients in the set $\{|\eta_k, \ldots, \eta_m|\}$. This good starting vector produces an iterate with dominant components in the subspace spanned by $u_k, \ldots, u_m$. Without advance knowledge of the eigenvectors, it is difficult to ensure the quality of a starting vector. This section concerns heuristics for starting vectors that work well in practice.

The difficulty of choosing starting vectors heuristically is demonstrated by a few simple examples. For instance, the canonical basis vectors e_1 and e_n should not be used as starting vectors because they are often nearly orthogonal to eigenvectors of a symmetric tridiagonal matrix T [75]. The vector of all ones is also a poor choice of starting vector as it is orthogonal to half of the eigenvectors of any symmetric tridiagonal Toeplitz matrix [36].

Analytic determination of a good starting vector is complicated by roundoff error in inverse iteration. As shown in [59, 75, 76, 77], one or two iterations in finite precision arithmetic are generally sufficient to produce a significant iterate component in the correct direction unless the starting vector is exactly orthogonal to that direction. In this section, the influence of the starting vector is assessed experimentally using the following test vectors:

1. $c =$ the "correct" eigenvector: For matrices [1,2,1] and W_g^+, accurate eigenvectors computed by inverse iteration were used. The starting vectors had residuals $\mathcal{R} < 10^{-14}$ for all orders and orthogonalities $\mathcal{O} < 10^{-14}$ for $n \leq 42$, $\mathcal{O} < 10^{-13}$ for $n \leq 105$, and $\mathcal{O} < 10^{-13}$ for $n \leq 525$. For the other test matrices, each correct eigenvector u_j is approximated by a different random vector.

2. $w + \tau c$: w is the computed eigenvector $\hat{u}_n$, and so is orthogonal to the eigenvectors corresponding to $\hat{\lambda}_1, \ldots, \hat{\lambda}_{n-1}$. A random starting vector is used in the computation of $\hat{u}_n$.

   Changing the value of τ shows how the rate of convergence depends on the size of the correct component. When random vectors are used instead of c, the experiments show how the rate of convergence depends on the size of a random perturbation away from the wrong (orthogonal) direction. These results give only an upper bound on the size of the correct component sufficient for rapid convergence.

3. random vectors: These vectors have uniformly distributed pseudorandom components between -1 and 1 generated using the linear congruential random number generator available from NETLIB. For each tested matrix order n, a single $n \times n$ random matrix is generated. Its columns are the starting vectors.

4. the TINVIT starting vector: This starting vector y is formed implicitly as suggested in [76] and shown in Algorithm 6.1.1 by factoring the shifted matrix $T - \lambda = LU$ and assuming that $Le = y$, where e is the vector of all ones.

Matrices with Distinct Eigenvalues

The first experiments use matrices with computationally distinct eigenvalues. Table 6.1 shows the number of inverse iterations required to compute the eigenvectors of the matrix [1,2,1] to the same accuracy achieved by TQL2 or TREEQL for each of the starting vector choices. The code used for inverse iteration is TINVIT modified to perform a fixed number of iterations. (The starting vectors and reorthogonalization criterion of TINVIT are retained.) High accuracy is achieved in one iteration only when accurately computed eigenvectors are used as starting vectors. More than two iterations are needed only when the starting vector is orthogonal to or nearly orthogonal to the computed eigenvector. All other tested starting vectors require two iterations. Thus, for matrix [1,2,1] a starting vector component η_j of magnitude $O(10^{-8})$ is sufficient for rapid computation of the eigenvector when the shift $\lambda = \hat{\lambda}_j$ and $n \leq 512$. Performing more iterations than the numbers listed in Table 6.1 does not significantly change the accuracy of the result. Specifically, as for TQL2 and TREEQL the minimum attainable orthogonality for all test problems seems strongly dependent on matrix order.

The results for matrix [1,2,1] were confirmed by experiments with fifty random matrices of order 100 and five random matrices of order 500. These test matrices have minimum eigenvalue spacing of about 10^{-4}. Only a few pairs of eigenvectors of [1,2,1] and of the random matrices are reorthogonalized. Table 6.2 shows the minimum, average, and maximum numbers of iterations required to attain full accuracy for each order. All of the tested starting vectors except those orthogonal to or nearly orthogonal to the solution lead to convergence in two iterations. A correct starting vector component of $O(10^{-8})$ is again sufficient for rapid convergence.

In summary, when all eigenvalues are distinct, the performance of inverse iteration is not strongly dependent on the starting vector unless the eigenvector is orthogonal to the starting vector. Random starting vectors and the TINVIT starting vectors provide the requisite component of $O(10^{-8})$ in the correct direction.

starting vector	$n = 32$ number of iterations for $\mathcal{R} < 10^{-14}$ $\mathcal{O} < 10^{-14}$	$n = 100$ number of iterations for $\mathcal{R} < 10^{-14}$ $\mathcal{O} < 10^{-13}$	$n = 512$ number of iterations for $\mathcal{R} < 10^{-14}$ $\mathcal{O} < 10^{-12}$
c	1	1	1
w	2	2	4
$w + 10^{-16}c$	2	2	3
$w + 10^{-8}c$	2	2	2
$w + 10^{-2}c$	2	2	2
same random	2	2	2
different random	2	2	2
TINVIT	2	2	2

Table 6.1: Number of inverse iterations per accurate eigenvector for matrix [1,2,1] of order n. Starting vector c is the correct computed eigenvector, and $w = \hat{u}_n$ The same number of iterations is performed for each eigenvector.

starting vector	n = 100 number of iterations for $\mathcal{R} < 10^{-14}$ $\mathcal{O} < 10^{-13}$ min	 avg	 max	n = 500 number of iterations for $\mathcal{R} < 10^{-14}$ $\mathcal{O} < 10^{-12}$ min	 avg	 max
w	3	4.9	5	> 5	> 5	> 5
$w + 10^{-16}c$	2	2.0	2	2	2.2	3
$w + 10^{-8}c$	2	2.0	2	2	2.0	2
$w + 10^{-2}c$	2	2.0	2	2	2.0	2
same random	2	2.0	2	2	2.0	2
different random	2	2.0	2	2	2.0	2
TINVIT	2	2.0	2	2	2.0	2

Table 6.2: Minimum, average, and maximum numbers of inverse iterations to compute accurate eigenvectors for fifty random matrices of order 100 and five random matrices of order 500. The matrices have minimum eigenvalue spacing 10^{-4}. The starting vector c is a different random vector for each computed eigenvector, and $w = \hat{u}_n$. The same number of iterations was performed for each eigenvector of a given matrix.

Matrices with Some Coincident Eigenvalues

When the computed eigenvalues $\hat{\lambda}_j$ and $\hat{\lambda}_{j+1}$ are equal, the eigenvector produced by the basic inverse iteration algorithm with $\lambda = \hat{\lambda}_j$ is the same as that found when $\lambda = \hat{\lambda}_{j+1}$. To overcome this difficulty, TINVIT employs a procedure suggested in [76]: the computationally coincident pair $\hat{\lambda}_j, \hat{\lambda}_{j+1}$ is replaced by $\hat{\lambda}_j, \hat{\lambda}_j + \epsilon$, where ϵ is on the order of machine precision times a norm of T. This substitution is intended to produce linearly independent eigenvectors from the two shifts without significantly increasing the residual error. Eigenvalue perturbation is used in the experiments described in this section for all starting vectors except the different random ones. When a different random vector is used for each eigenvalue, perturbing the eigenvalues does not change the experimental results.

Table 6.3 shows the results of the starting vector tests for the glued Wilkinson matrix W_g^+ with $n = 42$, 105, and 525. For the smallest orders, the results are nearly identical to those for matrix [1,2,1]. Thus, a component of $O(10^{-8})$ is again sufficient for rapid convergence. When $n = 525$, a correct component of $O(10^{-8})$ is still enough for fast convergence, but the number of iterations needed for accurate solution increases markedly over the $n = 42$ requirement for most of the other starting vectors. Specifically, with starting vectors orthogonal to the solution, a single random vector used for all starting vectors, and the TINVIT starting vectors, inverse iteration does not converge in the five iterations performed in this experiment.

Table 6.4 shows the smallest singular value of the matrices of first and second iterates before reorthogonalization. (The second iterates are computed from the reorthogonalized first iterates.) Except when different random starting vectors are used, the first iterates are linearly dependent, the modified Gram-Schmidt procedure fails, and near-zero vectors result. A second iteration also fails to produce linearly independent iterates for all but the random starting vectors. When the same random starting vector is used for all eigenvectors, the second set of iterates is linearly independent (the smallest computed singular value is not *exactly* zero) but of lesser quality than that produced from different random vectors. (See Table 6.3.)

While the implicitly generated TINVIT starting vectors are difficult to analyze, the other choices indicate a possible correlation between linearly dependent starting vectors and iterates: linearly dependent starting vectors lead to linearly dependent iterates in the case of computationally coincident eigenvalues. When a different random starting vector is used for each iterate, the results are linearly independent. Table 6.5 shows that the matrices with the n random starting vectors as columns have smallest singular value much larger than zero except when $n = 512$. In that case, the 512th column is linearly dependent on the first 479. As no test matrix has a cluster including both the 479th and the 512th eigenvalues, linearly independent starting vectors are used for all clustered eigenvalues for all tested matrix orders.

The same correlation between linear dependence of starting vectors and number of iterations can be seen to a lesser degree for other large ordered matrices with clustered eigenvalues. Table 6.6 shows the average number of iterations required to achieve high accuracy for fifty matrices of order 100 and for five matrices of order 500. When $n = 500$, using starting vector w leads to an average of over five iterations. The TINVIT starting vectors and a single random starting vector perform better for these matrices than for W_g^+, which has more and larger clusters than any of the other test matrices, but lead to as many as four iterations. Different random starting vectors are still the best heuristic choice with an average of 2.0 inverse iterations needed.

6.1.2 Stopping Criterion

In [75], Wilkinson shows that when the computed eigenvector z corresponding to the computed eigenvalue λ has a large norm before reorthogonalization, the eigenpair (λ, z) has a small residual. Specifically, if $\| z \|_2 > \epsilon_M / \kappa\sqrt{n}$, with $\epsilon_M =$ machine epsilon, and $y = \frac{z}{\|z\|_2}$, then $\| (T - \lambda)y \|_2 \leq 2\kappa\sqrt{n}\epsilon_M$, where κ

starting vector	$n = 42$ number of iterations for $\mathcal{R} < 10^{-14}$ $\mathcal{O} < 10^{-14}$	$n = 105$ number of iterations for $\mathcal{R} < 10^{-14}$ $\mathcal{O} < 10^{-13}$	$n = 525$ number of iterations for $\mathcal{R} < 10^{-14}$ $\mathcal{O} < 10^{-12}$
c	1	1	1
w	3	3	> 5
$w + 10^{-16}c$	3	3	> 5
$w + 10^{-8}c$	2	2	2
$w + 10^{-2}c$	2	2	2
same random	2	3	3
different random	2	2	2
TINVIT	2	3	> 5

Table 6.3: Number of inverse iterations required for high accuracy when the given starting vectors are used for the glued Wilkinson matrix W_g. Starting vector c is the correct computed eigenvector, and $w = \hat{u}_n$. The same number of iterations was performed for all eigenvectors of a given test matrix.

starting vector	smallest singular value of first iterates	minimum iterate norm after one iteration with $\min_j \| z_j \|_\infty$	smallest singular value of second iterates	minimum iterate norm after two iterations with $\min_j \| z_j \|_\infty$
w	0	0	0	$1.24d-13$
same random vector	0	$4.69d-12$	10^{-18}	$7.04d-04$
different random vector	0.02	$4.94d-04$	0.08	> 1.00
TINVIT	0	$4.94d-12$	0	$1.06d-12$

Table 6.4: Norm of the orthogonalized iterate after one and two iterations for the glued Wilkinson matrix W_g^+ of order 525 for four starting vector selections. The smallest singular value of the matrix of first iterates before reorthogonalization is also given. The starting vector is $w = \hat{u}_n$.

matrix order	smallest singular value
42	.0362
100	.0341
105	.0198
512	1.d-229
525	.0128

Table 6.5: The smallest singular value of the matrix of random starting vectors.

starting vector	n = 100 number of iterations for $\mathcal{R} < 10^{-14}$ $\mathcal{O} < 10^{-13}$			n = 500 number of iterations for $\mathcal{R} < 10^{-14}$ $\mathcal{O} < 10^{-12}$		
	min	avg	max	min	avg	max
w	2	3.2	4	4	> 5	> 5
$w + 10^{-16}c$	2	3.1	4	2	2.2	3
$w + 10^{-8}c$	2	2.2	3	2	2.0	2
$w + 10^{-2}c$	2	2.1	3	2	2.0	2
same random	2	2.1	3	2	2.8	4
different random	2	2.0	2	2	2.0	2
TINVIT	2	2.8	4	2	3.0	4

Table 6.6: Minimum, average, and maximum numbers of inverse iterations required for high accuracy when the given starting vectors are used for fifty random matrices of order 100 and five random matrices of order 500. All matrices have some clustered eigenvalues. The starting vector c is a different random vector for each computed eigenvector, and $w = \hat{u}_n$.

is a constant of order unity depending on the type of arithmetic used. When λ is a good approximation to a distinct eigenvalue λ_j of T, a large norm $\| z \|_2$ signals a large component in the u_j direction. When λ approximates a cluster of eigenvalues $\lambda_k, \ldots, \lambda_m$ a large norm means that the iterate approximates a linear combination of the eigenvectors $u_k, \ldots, u_m$. In the latter case, the iterates corresponding to the clustered eigenvalues are reorthogonalized to produce an orthogonal basis for the subspace spanned by $u_k, \ldots, u_m$. The meaning of a small norm for a reorthogonalized iterate was discussed in Section 6.1.1. The inequality $\| z \|_2 \geq \| z \|_\infty$ shows that the less expensive infinity norm can be used as stopping criterion in the implementation of inverse iteration: when $\hat{\lambda}_k$ is distinct from $\hat{\lambda}_{k-1}$ and $\hat{\lambda}_{k+1}$, $\| z \|_\infty > \epsilon_M / \kappa\sqrt{n}$ signals a small residual.

As noted in [75], the norm is probably the vector property most closely correlated with iterate quality. The difficulty lies in quantifying that correlation. For example, in TINVIT, iteration stops when the iterate (scaled by a factor of $O(\epsilon_M)$ to prevent overflow) has infinity norm larger than one. If reorthogonalization is needed, the norm is calculated after the iterate is reorthogonalized (that is, after step 3.b in Algorithm 6.1.1 and before the iterate is normalized). The experimental results given in this section show that TINVIT's choice causes inverse iteration to stop before highest accuracy is attained. An alternative is suggested that consistently improves the accuracy of inverse iteration.

Table 6.7 shows how the accuracy of the computed eigendecomposition depends on the norm of the computed iterates (after reorthogonalization, if performed, and before normalization) for matrices [1,2,1] and W_g^+. The TINVIT stopping criterion works correctly for both orders of the glued Wilkinson matrix W_g^+: unit iterate norm and full accuracy are both attained on the second iteration. It fails, however, on the matrix [1,2,1] where all iterates have greater than unit norm but less than full accuracy on the first iteration. These results suggest that at least two iterations should always be performed regardless of iterate norm when different random starting vectors are used.

Similar tests for 50 matrices of order 100 and five matrices of order 500 with some clustered eigenvalues indicate that a somewhat stronger criterion is, in fact, in order. Tables 6.8 and 6.9 show maximum residuals $\mathcal{R}$ and orthogonalities $\mathcal{O}$ taken over the tested matrices. The first values of $\mathcal{R}$ and $\mathcal{O}$ were measured after the first iteration where all iterates have norm greater than one; the second values of $\mathcal{R}$ and $\mathcal{O}$ were measured after the second iteration where all iterates have norm greater than one. For all tested matrices, these were consecutive iterations. For some of the matrices, they were the first and second iterations performed. As full accuracy is not achieved, on average, until two iterations with unit norms have been completed, these data suggest that at least that many iterations should be performed. There does not appear to be any simple correlation between cluster size and iterate norm.

6.1.3 Reorthogonalization

The reorthogonalization step is necessary whenever close eigenvalues occur. It remains only to determine when eigenvalues should be considered close. Suppose that the matrix T has diagonal elements $\alpha_1, \ldots, \alpha_n$ and off-diagonal elements $\beta_2, \ldots, \beta_n$. TINVIT uses a reorthogonalization criterion of $10^{-3}\| T \|_R$. That is, if a computed eigenvalue $\hat{\lambda}_j$ is separated by a distance less than $10^{-3}\| T \|_R$ from the next largest eigenvalue $\hat{\lambda}_{j-1}$, the computed eigenvector $\hat{u}_j$ is reorthogonalized against $\hat{u}_{j-1}$ and all eigenvectors against which $\hat{u}_{j-1}$ was orthogonalized.

Table 6.10 shows the residuals $\mathcal{R}$ and orthogonalities $\mathcal{O}$ for the matrix [1,2,1] of order 100 as the reorthogonalization criterion is varied from 0 to $10^{10}\| T \|_R$ after one and two inverse iterations. A different random starting vector is used for each eigenvector computation. Reorthogonalization takes place for the same eigenvectors at both iterations. This table shows that additional orthogonalization is not a substitute for extra iterations because small residuals are not attained until the second iteration.

Table 6.11 shows the same data after two iterations for W_g^+ when $n = 105$ and $n = 525$. In addition, the fraction of inverse iteration time spent in the modified Gram-Schmidt procedure is shown. For these matrices, increasing the grouping tolerance beyond that of TINVIT does not improve the accuracy of the

matrix	iteration	minimum iterate norm $\min_j \parallel z_j \parallel_\infty$	residual $\mathcal{R}$	orthogonality $\mathcal{O}$
$[1,2,1]$ $n = 100$	1	> 1.00	$3.18d-14$	$3.05d-12$
	2	> 1.00	$1.58d-16$	$3.20d-14$
W_g^+ $n = 105$	1	0.14	$1.47d-13$	$1.68d-11$
	2	> 1.00	$8.70d-16$	$3.85d-15$
$[1,2,1]$ $n = 512$	1	> 1.00	$1.60d-11$	$4.62d-09$
	2	> 1.00	$3.93d-16$	$1.57d-13$
W_g^+ $n = 525$	1	$4.94d-04$	$3.42d-09$	$6.02d-07$
	2	> 1.00	$5.99d-15$	$1.98d-14$

Table 6.7: Iterate norm, residual, and orthogonality for matrices [1,2,1] and W_g^+ after one and two iterations. A different random starting vector is used for each eigenvector computation.

iteration	maximum residual $\mathcal{R}$	maximum orthogonality $\mathcal{O}$
first iteration with $\| z_j \|_\infty > 1$ for $j = 1, \ldots, 100$.	$2.47d - 12$	$5.14d - 11$
second iteration with $\| z_j \|_\infty > 1$ for $j = 1, \ldots, 100$.	$3.35d - 16$	$1.10d - 14$

Table 6.8: Maximum residual and orthogonality for 50 test matrices of order 100 with clustered eigenvalues. Results are given for the first iteration at which all iterate norms are greater than one and for the subsequent iteration. A different random starting vector is used for each eigenvector computation.

iteration	maximum residual $\mathcal{R}$	maximum orthogonality $\mathcal{O}$
first iteration with $\| z_j \|_\infty > 1$ for $j = 1, \ldots, 500$.	$1.19d - 12$	$7.01d - 11$
second iteration with $\| z_j \|_\infty > 1$ for $j = 1, \ldots, 500$.	$8.70d - 16$	$7.10d - 14$

Table 6.9: Maximum residual and orthogonality for 5 test matrices of order 500 with clustered eigenvalues. Results are given for the first iteration at which all iterate norms are greater than one and for the subsequent iteration. A different random starting vector is used for each eigenvector computation.

criterion	number of vectors orthogonalized	one iteration		two iterations	
		$\mathcal{R}$	$\mathcal{O}$	$\mathcal{R}$	$\mathcal{O}$
$10^{10}\| T \|_R$ (all)	99	6.09d-13	6.41d-15	2.12d-16	5.75d-15
$10^{-1}\| T \|_R$	97	1.87d-12	7.40d-15	2.13d-16	6.12d-15
$10^{-2}\| T \|_R$	33	7.69d-14	4.97d-12	1.67d-16	1.82d-15
$10^{-3}\| T \|_R$	2	3.18d-13	3.05d-12	1.58d-16	3.20d-14
$10^{-5}\| T \|_R$	1	1.10d-13	2.75d-11	1.90d-16	4.28d-14
0	0	2.28d-13	2.68d-11	1.61d-16	4.68d-14

Table 6.10: Variation of accuracy and reorthogonalization time with reorthogonalization criterion for matrix [1,2,1] when $n = 100$.

result. It does, however, substantially increase the time required for the solution. When the TINVIT criterion is used, most of the eigenvectors are reorthogonalized (80% when $n = 105$ and 96% when $n = 525$), but reorthogonalization occurs in many small clusters. Thus, when all eigenvectors are reorthogonalized as one group, the cost rises markedly although the accuracy is little changed. The same sort of increase is seen, on the average for the fifty matrices of order 100 with clustered eigenvalues. Table 6.12 shows that the TINVIT criterion provides the desired accuracy. Reorthogonalizing all vectors improves the accuracy slightly with a large increase in computing time. Performing more iterations does not significantly change the accuracy of the results in Tables 6.10–6.12.

These experiments show that the best possible orthogonality can be guaranteed in general only by the time-consuming process of reorthogonalizing all eigenvectors. The desired high accuracy, however, can usually be achieved using the TINVIT reorthogonalization criterion along with random starting vectors and the improved stopping criterion derived in Section 6.1.3.

6.2 A New Implementation of Inverse Iteration

The improvements to inverse iteration developed in this chapter are incorporated into the following algorithm. Note that the heuristics chosen are based on experiments with matrix orders up to about 500 and may not apply to much larger matrix orders.

Algorithm 6.2.1 (Improved Inverse Iteration Algorithm (III))

For $j = 1, \ldots, n$

1. *Initialize the set of eigenvalues close to* $\hat{\lambda}_j$*: CLUSTER*$(\hat{\lambda}_j) = \emptyset$.

   If $j > 1$ *and* $|\hat{\lambda}_j - \hat{\lambda}_{j-1}| < 10^{-3}\| T \|_R$*, then*

$$\begin{aligned} CLUSTER(\hat{\lambda}_j) &= CLUSTER(\hat{\lambda}_{j-1}) \bigcup \{\hat{\lambda}_{j-1}\} \\ &= \{\hat{\lambda}_i, \ldots, \hat{\lambda}_{j-1}\}, \quad i \leq j-1. \end{aligned}$$

order	criterion	$\mathcal{R}$	$\mathcal{O}$	number of vectors	time for MGS	fraction MGS time
$n = 105$	$10^{10} \| T \|_R$ (all)	1.97d-16	4.57d-15	104	30.6	.67
	$10^{-1} \| T \|_R$	1.60d-16	3.14d-15	88	27.7	.10
	$10^{-3} \| T \|_R$	8.70d-16	3.85d-15	84	1.4	.07
	$10^{-5} \| T \|_R$	2.09d-16	2.52d-13	78	1.4	.05
	0	1.85d-16	2.05	0	0	0
$n = 525$	$10^{10} \| T \|_R$ (all)	7.58d-15	1.53d-14	524	5807.1	.98
	$10^{-1} \| T \|_R$	4.69d-15	3.51d-14	523	5287.6	.73
	$10^{-3} \| T \|_R$	5.99d-15	1.98d-14	504	338.1	.15
	$10^{-5} \| T \|_R$	3.46d-15	6.24d-13	498	253.1	.12
	0	2.04d-16	6.38	0	0	0

Table 6.11: Variation of accuracy and time after two inverse iterations with reorthogonalization criterion for matrix W_g^+ when $n = 100$ and $n = 525$. The last column shows the fraction of inverse iteration time spent in reorthogonalization.

reorthogonalization criterion	average $\mathcal{R}$	average $\mathcal{O}$	average fraction of eigenvectors	fraction MGS time
$10^{10} \| T \|_R$	4.98d-16	7.42d-15	49.4	0.70
$10^{-3} \| T \|_R$	4.18d-16	4.13d-15	5.7	0.02

Table 6.12: Average variation of accuracy and reorthogonalization time with reorthogonalization criterion for fifty matrices of order 100 with some clustered eigenvalues after two iterations.

```
2. Generate a random vector $x_j$ with uniformly distributed components in the
   interval [-1,1], and form the starting vector $y_j = \frac{x_j}{\|x_j\|_2}$.
3. Set $\sigma = 0$.
4. Loop until the iterate norm $\sigma \geq 1.0$. (Error exit after 5 iterations.)
    4.a Solve tridiagonal system $(T - \hat{\lambda}_j)z_j = y_j$.
    4.b Reorthogonalize $z_j$ with respect to $\hat{u}_i, \ldots, \hat{u}_{j-1}$.
    4.c Set $\sigma = \| z_j \|_\infty$.
5. Repeat steps 3.a and 3.b once.
6. Accept $\frac{z_j}{\|z_j\|_2}$ as computed eigenvector $\hat{u}_j$.
```

In Algorithm III, the number of iterations performed is determined independently for each eigenvector. Because convergence cannot be guaranteed, iteration is discontinued and an error signalled if five iterations are performed. In the implementation tested in this section, the iterates are scaled to prevent overflow as in TINVIT. The storage requirements are the same for this implementation as for TINVIT, and parallel implementation proceeds as described for TINVIT in Chapter 4.

n		TSTURM			BISECT / III		
	time to compute eigenvalues (seconds)	time to compute eigenvectors (seconds)	$\mathcal{R}$	$\mathcal{O}$	time to compute eigenvectors (seconds)	$\mathcal{R}$	$\mathcal{O}$
32	1.1	0.3	4.15d-15	4.00d-13	0.4	1.30d-16	4.27d-15
100	11.3	2.0	2.46d-14	8.48d-12	3.2	1.56d-16	3.15d-14
512	276.7	72.2	1.26d-13	4.48d-11	125.8	4.11d-16	1.78d-13

Table 6.13: Times, residuals, and orthogonalities for eigensystems computed by TSTURM and by BISECT with III for matrix $[1, 2, 1]$.

n		TSTURM			BISECT / III		
	time to compute eigenvalues (seconds)	time to compute eigenvectors (seconds)	$\mathcal{R}$	$\mathcal{O}$	time to compute eigenvectors (seconds)	$\mathcal{R}$	$\mathcal{O}$
42	1.6	0.4	4.25d-15	2.3d-13	0.6	1.61d-16	2.61d-15
105	4.72	3.0	5.11d-14	2.36d-12	4.8	6.98d-16	4.43d-15
525	23.3	171.1	1.14d-13	4.08d-11	333.4	5.55d-15	1.69d-14

Table 6.14: Times, residuals, and orthogonalities for eigensystems computed by TSTURM and by BISECT with III for matrix W_g^+.

Tables 6.13 and 6.14 compare the costs for eigenvector computation by the improved algorithm and TSTURM. Because additional iterations are performed in the new implementation, its cost is substantially higher than that of the eigenvector computation in TSTURM. For matrix [1,2,1], where eigenvector computation is cheap compared to eigenvalue computation, however, the longer time represents only a 13% increase in total computation time when $n = 512$.

In Algorithm III, the time for generation of random starting vectors is small compared to the total computation time. A total of 10^6 random vector elements (a 1000×1000 matrix) can be generated in 14.00 seconds on a Sequent Symmetry S81 processor with an 1167 floating point accelerator. The time to generate random starting vectors comprises less than 4% of the total eigenvector computation time for matrix [1,2,1] of orders up to 512 and for W_g^+ for orders up to 525.

6.3 A Serial Comparison of TREEQL, TQL2, and B/III

This section offers an experimental comparison of Cuppen's divide and conquer method, the QL method, and bisection with inverse iteration. The respective implementations are TREEQL [24], TQL2 [68], and BISECT with III (B/III). All experiments were performed in double precision on a single Sequent processor using the Weitek 1167 floating-point accelerator. The results show that all three codes produce highly accurate eigendecompositions. Data are presented for test matrices [1,2,1], W_g^+, and [1,u,1] as they illustrate the range of results for all test matrices from Chapter 2.

Table 6.16 shows the maximum residual $\mathcal{R} = \max_j \| Tu_j - \lambda_j u_j \|_2$ and orthogonality $\mathcal{O} = \| U^T U - I \|_\infty$ of eigenvalues and eigenvectors computed by B/III, TREEQL, and TQL2 for the test problems. All methods solve the problems to roughly the same high degree of accuracy.

The relative speeds of the methods for a given problem are determined by the degree of deflation in TREEQL, the amount of matrix splitting in TQL2,

matrix	order n	TREEQL: roots computed (scaled)	TQL2: $\mathcal{N} = \frac{1}{n}\sum_{i=1}^{m} n_i$	B/III: fraction of BISECT time
$[1,2,1]$	32	2.2	39	.76
	100	2.4	114	.75
	512	2.2	502	.66
W_g^+	42	2.4	45	.76
	105	1.3	111	.60
	512	0.3	504	.08
$[1,\mu,1]$	32	3.0	40	.78
	100	3.0	112	.76
	512	3.0	455	.74

Table 6.15: The number of roots computed by TREEQL divided by the matrix order, the order index for TQL2, and the fraction of time spent in BISECT by B/III for matrices [1,2,1], W_g^+, and [1,u,1].

and the clustering of eigenvalues. The first of these quantities is measured by the total number of roots computed by TREEQL divided by the matrix order n. For the problems solved, the TREEQL computational tree has four levels, and n eigenpairs are determined at each level. The problems at the lowest level are solved by TQL2 and are not included in the count. Thus, when no deflation occurs, the scaled number of roots is $3n/n = 3$.

The amount of matrix splitting in TQL2 is measured by $\mathcal{N} = \frac{1}{n}\sum_{i=1}^{m} n_i$, where n_i is the submatrix order at iteration i, and m is the total number of iterations. Table 6.15 suggests that eigenvalue clustering (with cluster size much less than matrix order) reduces both the total runtime of B/III and the ratio of BISECT to III time. These results are confirmed by the data in Table 6.17 which shows the total number of clustered eigenvalues, the maximum cluster size, the total time for B/III, and the fractions of that total contributed by BISECT and by reorthogonalization by the modified Gram-Schmidt process. An eigenvalue is counted as part of a cluster if $\hat{\lambda}_j - \hat{\lambda}_{j+1} < 10^{-14}\| T \|_2$, so a single cluster of two eigenvalues is listed as one clustered eigenvalue.

matrix order	method	maximum residual $\mathcal{R}$	maximum orthogonality $\mathcal{O}$
n = 32 or 42	TREEQL	3.26d-15	5.59d-15
	TQL2	1.52d-15	1.30d-14
	B/III	1.80d-16	6.20d-15
n = 100 or 105	TREEQL	6.07d-14	2.75d-15
	TQL2	2.39d-15	1.06d-14
	B/III	3.67d-15	8.52d-14
n = 512 or 525	TREEQL	3.96d-15	1.67d-13
	TQL2	1.66d-14	2.50d-13
	B/III	6.05d-15	7.92d-13

Table 6.16: Maximum residual and orthogonalities of eigendecompositions computed by B/III, TREEQL, and TQL2 for the three test matrices.

matrix	order	number of clustered eigenvalues	maximum cluster size	time for B/III (seconds)	time ratios	
					$\frac{\textsc{Bisect}}{\textsc{Total}}$	$\frac{\textsc{Mgs}}{\textsc{Total}}$
$[1,2,1]$	32	0	0	1.48	.76	0
	100	0	0	13.38	.75	0
	200	0	0	52.19	.75	0
	512	0	0	372.70	.66	.11
W^+	33	5	2	1.30	.69	.04
	101	32	2	9.41	.66	.03
	201	73	2	35.24	.63	.03
	513	82	2	215.61	.62	.03
random	32	0	0	1.46	.77	0
	100	0	0	13.27	.77	0
	200	0	0	51.68	.76	0
	512	0	0	329.83	.75	.01
W_g^+	42	4	2	1.43	.76	0
	105	53	5	9.83	.60	.09
	210	158	10	34.09	.32	.32
	525	473	25	334.68	.08	.68
modified $[1,2,1]$	32	1	2	1.43	.75	0
	100	3	2	12.96	.75	0
	200	1	2	51.87	.75	.01
	512	1	2	369.20	.67	.11
$[1,u,1]$	32	0	0	1.50	.78	0
	100	0	0	13.55	.78	0
	200	0	0	52.82	.76	0
	512	0	0	340.02	.74	.01

Table 6.17: Number of singular values with spacing less than $10^{-14}\| T \|_2$, maximum cluster size, and fractions of B/III times spent in BISECT and MGS.

As the number of clustered eigenvalues grows, the fraction of time spent in BISECT shrinks while the fraction spent in reorthogonalization grows. This happens because clustered eigenvalues are computed quickly by bisection but their eigenvectors must be reorthogonalized. For small clusters (W^+), the total is lowered because the cost for reorthogonalization does not outweigh the savings in bisection. For large clusters (W_g^+), the significant cost of reorthogonalization brings the total cost to that of nonclustered problems.

Table 6.15 compares the other measures for the test matrices. The root counts for TREEQL show that, for large orders, matrix [1,2,1] undergoes a moderate amount of deflation, matrix [1,u,1] very little deflation, and W_g^+ a great deal of deflation. The values of $\mathcal{N}$ reveal that all matrices of the same order undergo about the same amount of matrix splitting with TQL2. These properties are reflected in the runtimes of the three methods.

The top graphs in Figures 6.1–6.3 show the time required to solve the test eigenproblems of orders up to 60 by B/III, TREEQL, and TQL2. The same data are given in Table 6.18. At orders less than 20, B/III is slowest and TQL2 fastest for matrices [1,2,1] and [1,u,1]. For moderate ([1,2,1]) to heavy (W_g^+) deflation, TREEQL is the fastest method for orders 20 to 60. Because of its extremely high degree of deflation, TREEQL is always the fastest method for W_g^+. For matrices of about order 50 and higher, TQL2 becomes the slowest technique. (TREEQL switches from divide and conquer to TQL2 to solve subproblems of order 50 or less.) The bottom graphs in Figures 6.1–6.3 show the time for the three methods for matrix orders up to 512. For orders larger than about 40, the speeds are largely dependent on how much deflation occurs during the solution of the eigenproblem by TREEQL. For orders 512 and 525, TREEQL is about 2 to 40 times faster than TQL2. For these orders, B/III is consistently about eight times faster than TQL2.

Because the degree of deflation and the clustering of eigenvalues are not readily determined without solution of the problem, it is not generally possible to determine in advance the fastest serial method for the symmetric tridiagonal eigenproblem. However, for all tested matrices, the combination B/III is much faster than TQL2 and equally accurate. For large orders and zero to medium deflation, B/III is fastest, while for large amounts of deflation, TREEQL is fastest.

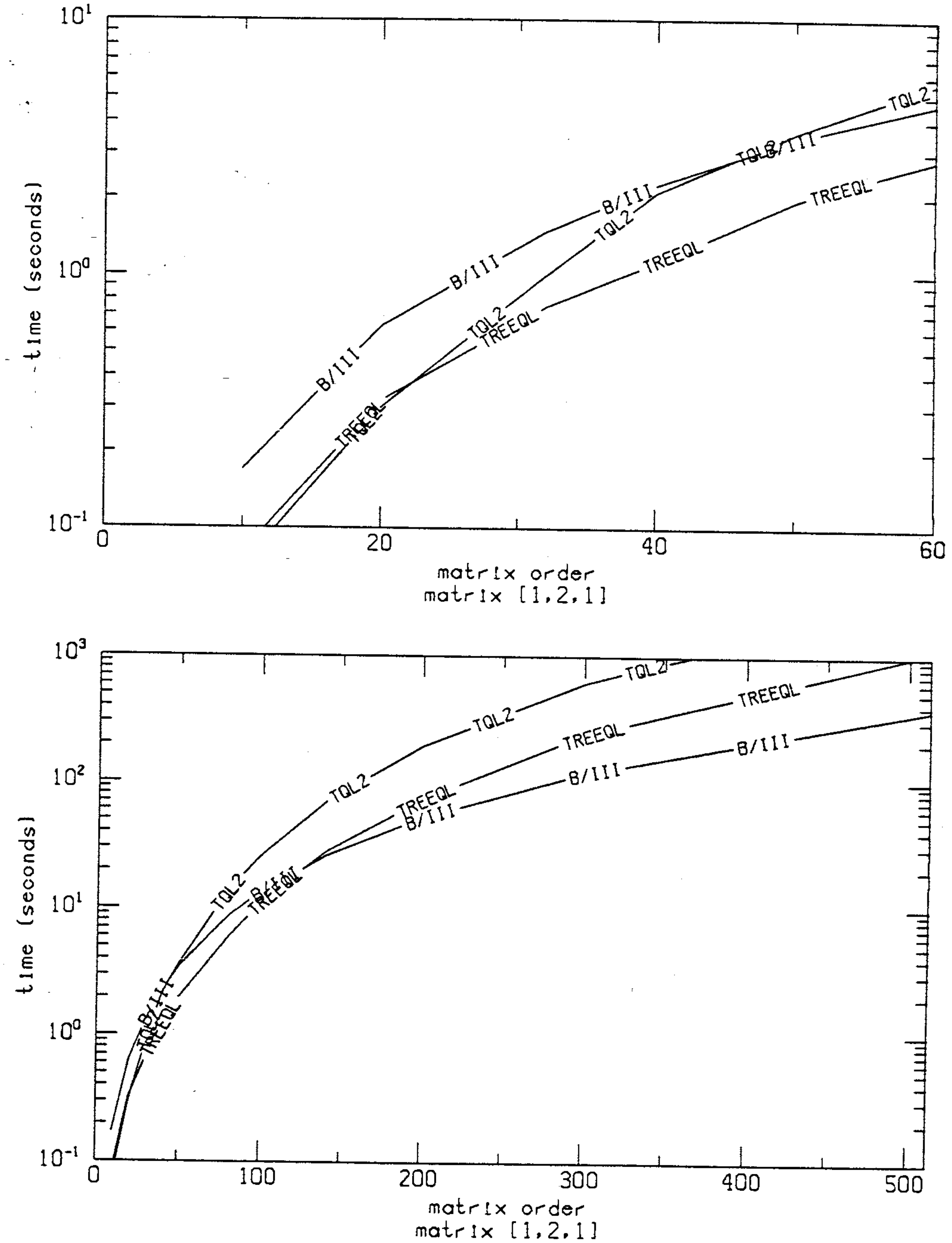

Figure 6.1: Times for TQL2, TREEQL, and B/III *versus* matrix order for matrix [1,2,1].

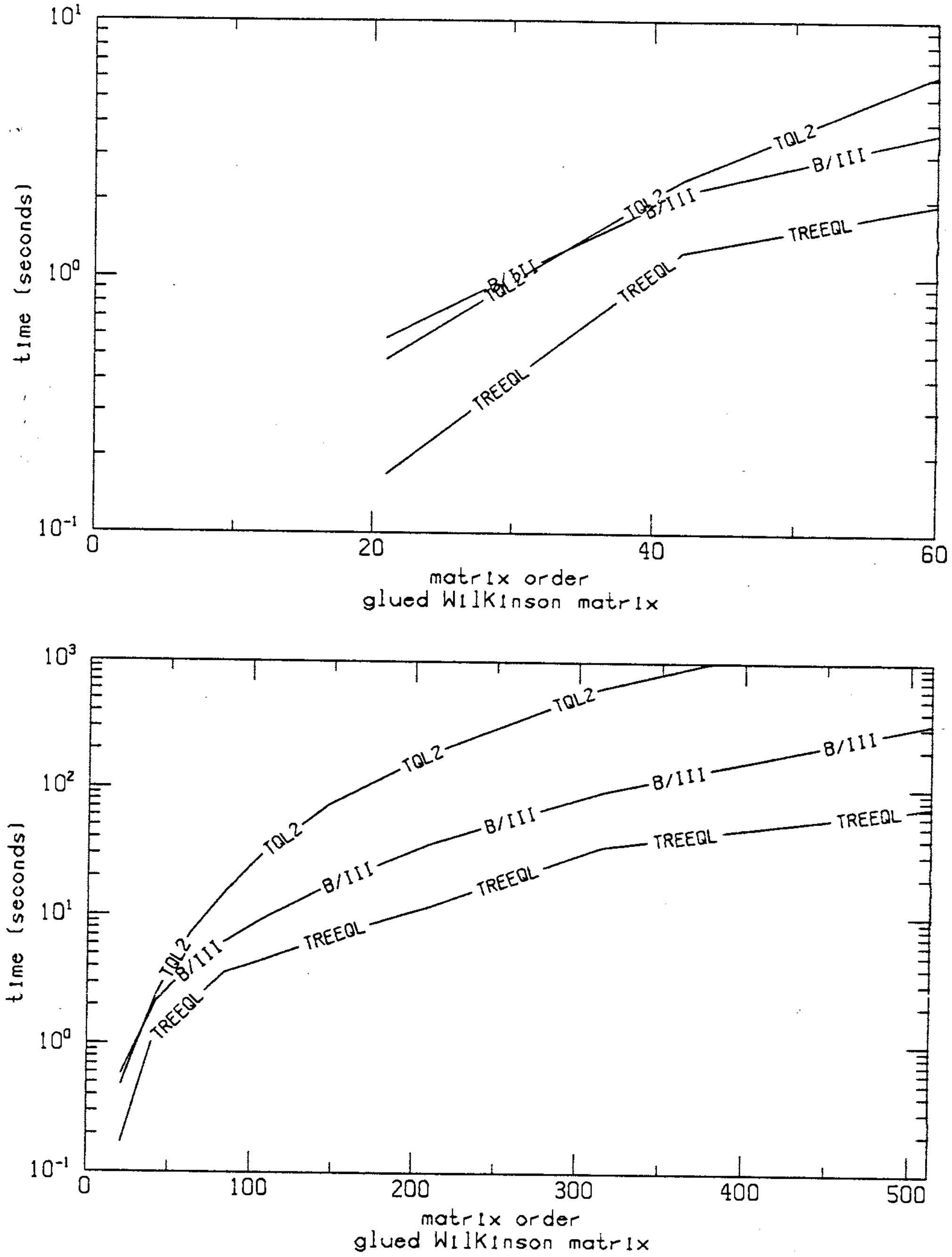

Figure 6.2: Times for TQL2, TREEQL, and B/III *versus* matrix order for the glued Wilkinson matrix W_g^+.

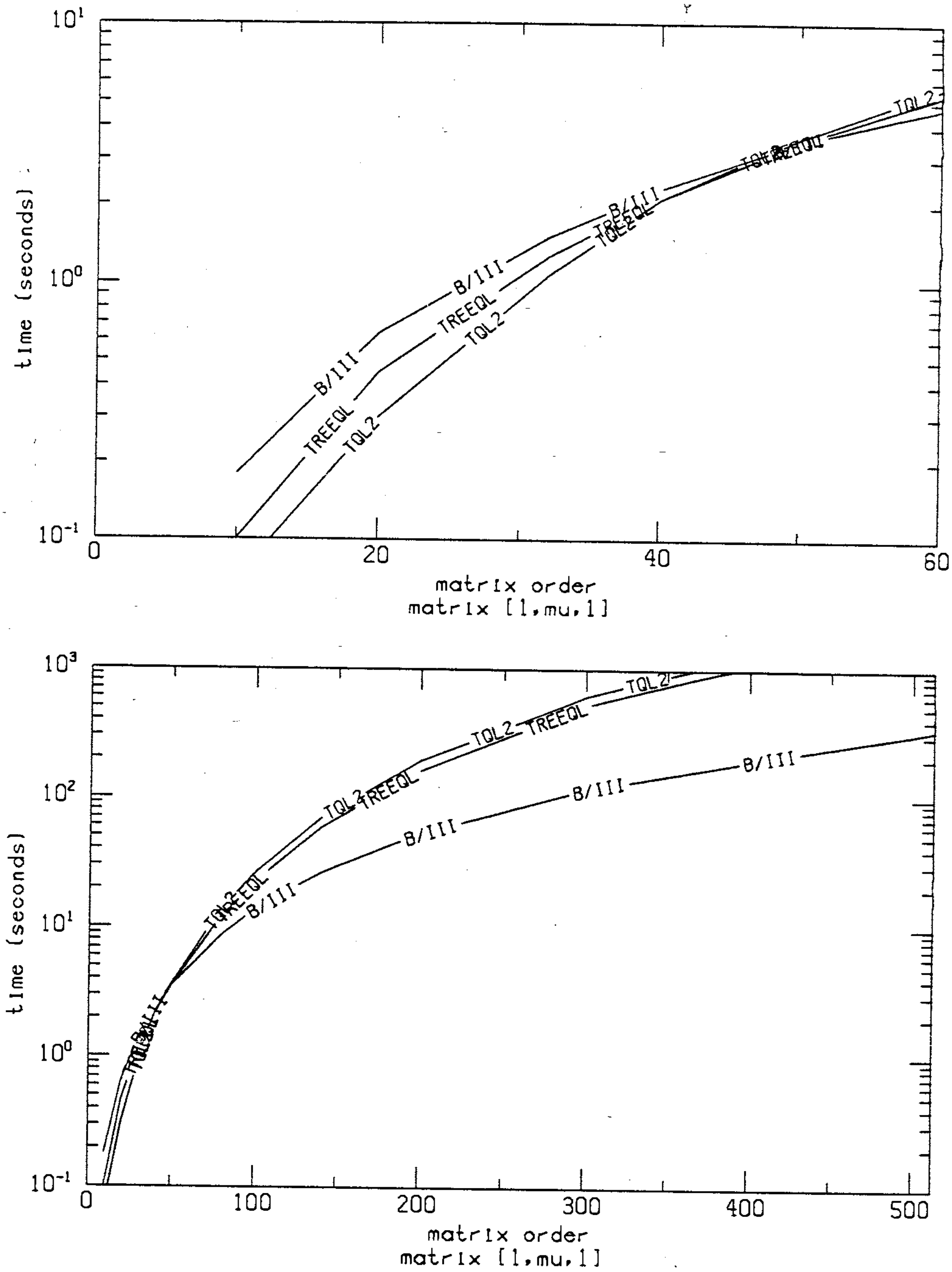

Figure 6.3: Times for TQL2, TREEQL, and B/III *versus* matrix order for matrix [1,u,1].

matrix	order	time for TQL2 (seconds)	time for TREEQL (seconds)	time for B/III (seconds)
$[1,2,1]$	10	.07	.08	.17
	20	.30	.32	.63
	32	1.00	.75	1.48
	40	2.10	1.12	2.25
	50	3.60	1.95	3.43
	80	13.73	5.85	8.63
	100	25.93	11.20	13.38
	140	68.27	27.88	25.80
	200	190.40	72.33	52.18
	300	615.67	240.05	117.96
	512	2965.03	1084.11	372.70
W_g^+	21	.48	.17	.58
	42	2.40	1.25	2.13
	63	7.23	2.08	4.03
	84	5.25	3.65	6.30
	105	8.48	4.42	9.13
	147	72.42	6.90	17.00
	210	87.47	11.87	36.28
	315	629.58	35.10	94.32
	525	2741.25	75.77	355.92
$[1,\mu,1]$	10	.07	.10	.18
	20	.30	.45	.64
	32	1.07	1.27	1.5
	40	2.10	2.10	2.30
	50	3.65	3.53	3.53
	80	13.80	12.10	8.73
	100	26.30	22.50	13.55
	140	68.90	58.20	26.18
	200	191.50	160.40	52.82
	300	617.12	520.53	117.47
	512	2983.30	2491.85	340.02

Table 6.18: Times for TQL2, TREEQL, and B/III for [1,2,1], W_g^+, and [1,u,1].

Chapter 7

A Statistical Analysis of Inverse Iteration

Chapter 6 establishes the design choices needed for an accurate implementation of inverse iteration. These rely in large part on the use of starting vectors with randomly distributed components. In this chapter, statistical analysis is used to explain some of the experimental observations.

The first three sections of this chapter lay the foundations for the analysis. Section 7.1 states the underlying assumptions. Section 7.2 presents geometric and analytic definitions of a good eigenvector approximation. The analytic formulation depends on evaluating the incomplete beta function $\mathcal{I}_{\tau^2}$, and Section 7.3 proves a tight upper bound for $\mathcal{I}_{\tau^2}$.

The application of statistics is discussed in the remaining sections. Section 7.4 determines the expected quality of a random starting vector. Sections 7.5 and 7.6 discuss why statistical analysis can only be applied in a limited fashion to the iterates before and after reorthogonalization. Section 7.7 estimates the error in applying the statistical analysis of starting vectors with normally distributed components to the starting vectors with uniformly distributed components used in the experiments. The statistical fundamentals required for the proofs of the theorems in this chapter are reviewed in Section 7.8.

7.1 Assumptions

For this analysis of inverse iteration, it is assumed that the starting vector is created in the following way. A vector x_0 is first generated with independent random components each having a normal distribution with mean 0 and variance 1 (normal (0,1)). This vector is then normalized to give the starting vector $y_0 = x_0 / \| x_0 \|_2$ having unit norm. Vectors so formed are uniformly distributed on the unit n-sphere [20]. Unless otherwise specified, all vectors are represented in terms of the orthonormal basis of eigenvectors $\{u_1, u_2, \ldots, u_n\}$ of the symmetric tridiagonal matrix T, *i.e.*, $y_0 = (\eta_1, \eta_2, \ldots, \eta_n)^T$ means $y_0 = \sum_{i=1}^n \eta_i u_i$ and $\| u_i \|_2 = 1$. Use of the basis of eigenvectors is permitted because the distribution of the components of vectors uniformly distributed on the sphere is invariant under orthogonal transformations. This is shown algebraically in the following theorem.

Theorem 7.1.1 *Let $v = (\xi_1, \ldots, \xi_n)^T$, where the components ξ_i are independent random variates with normal (0,1) distributions and denote coordinates with respect to an arbitrary basis. If $U = (u_1, \ldots, u_n)$ is an $n \times n$ orthogonal matrix, then Uv has components $\nu_1, \ldots, \nu_n$ distributed identically to those of v.*

Proof: As a normal (0,1) variate, ξ_i has characteristic function $\phi_i(t) = e^{-t^2/2}$ for $i = 1, \ldots, n$. According to Lemma 7.8.2, the characteristic function of $\nu_i = \sum_{j=1}^n u_{ij}\xi_j$, $1 \le i \le n$, is

$$\phi(t) = \prod_{j=1}^{n} \phi_j(u_{ij}t) = \prod_{j=1}^{n} e^{-u_{ij}^2 t^2/2} = e^{\frac{-t^2}{2}\sum_{j=1}^n u_{ij}^2} = e^{\frac{-t^2}{2}}$$

because $\sum_{j=1}^n u_{ij}^2 = 1$ for an orthogonal matrix. ∎

7.2 The Quality of an Approximate Vector

To carry out the statistical analysis, it is necessary to first determine what is meant by a "good" approximation to an eigenvector or to a linear combination of eigenvectors. Equivalent geometric and analytic definitions follow along with examples in three dimensions.

Let $y_0 = \sum_{i=1}^{n} \eta_i u_i$ satisfy $\| y_0 \|_2 = 1$. y_0 is considered a good approximation to u_i if η_i is much larger than any other component, *i.e.*, if $\eta_i^2 \geq 1 - \epsilon^2$ for some tiny error tolerance ϵ. Similarly, y_0 is nearly a linear combination of eigenvectors $u_1, \ldots, u_d$ if $\sum_{i=1}^{d} \eta_i^2 \geq 1 - \epsilon^2$. In geometric terms, a good approximation y_0 to u_i makes an angle θ_i with u_i having $\cos\theta_i \geq \sqrt{1-\epsilon^2}$. Because random vectors are uniformly distributed on the sphere, the probability that a random starting vector is a good approximation to such a linear combination is just the fraction of the surface area of the sphere defined by good vectors.

In analytic terms, the probability that $\sum_{i=1}^{d} \eta_i^2 \geq 1 - \epsilon^2$ is determined by integrating the probability density function of the sum $\sum_{i=1}^{d} \eta_i^2$ between $1 - \epsilon^2$ and 1. The component ξ_i has a normal (0,1) distribution. Therefore, the term $\eta_i^2 = \frac{\xi_i^2}{\sum_{j=1}^{n} \xi_j^2}$ has a $B(\frac{1}{2}, \frac{n-1}{2})$ distribution, and the sum $\sum_{i=1}^{d} \eta_i^2$ has a $\mathrm{B}(\frac{\mathrm{d}}{2}, \frac{\mathrm{n-d}}{2})$ distribution [20]. The beta distribution is described in Section 7.8. The probability that y_0 closely approximates a linear combination of the eigenvectors $u_1, \ldots, u_d$ is then

$$
\begin{aligned}
P(\sum_{i=1}^{d} \eta_i^2 \geq 1 - \epsilon^2) &= \alpha \int_{1-\epsilon^2}^{1} t^{\frac{d}{2}-1}(1-t)^{\frac{n-d-2}{2}} dt \\
&= 1 - \alpha \int_{0}^{1-\epsilon^2} t^{\frac{d}{2}-1}(1-t)^{\frac{n-d-2}{2}} dt, \qquad (7.1)
\end{aligned}
$$

with $\alpha = \frac{\gamma(\frac{n}{2})}{\gamma(\frac{d}{2})\gamma(\frac{n-d}{2})}$.

These geometric and algebraic definitions of a good vector approximation are readily illustrated in three dimensions. When $n = 3$ and $d = 1$ or $d = 2$, $\alpha = \frac{\Gamma(3/2)}{\Gamma(1/2)\Gamma(1)} = \frac{1}{2}$. Using this value and equation (7.1), the probability that y_0 is a good approximation to u_3 (or $-u_3$) is

$$
P(\eta_3^2 \geq 1 - \epsilon^2) \geq 1 - \frac{1}{2} \int_{0}^{1-\epsilon^2} t^{-\frac{1}{2}}(1-t)^0 dt.
$$

In this case, $P(\eta_3^2 \geq 1 - \epsilon^2)$ is the probability that y_0 lies inside the double-sided cone around u_3 having vertex at the center of the unit sphere and making interior

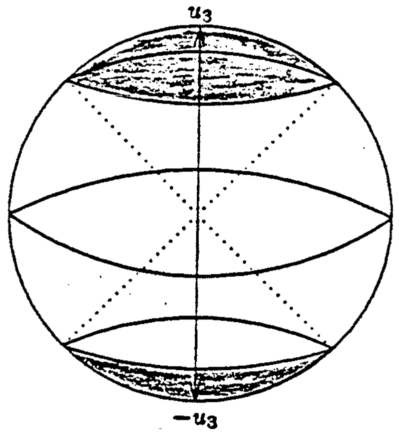

Figure 7.1: Vectors on the Unit 3-Sphere with $\eta_3^2 \geq 1 - \epsilon^2$

angle $\cos^{-1} \eta_3$ with eigenvector u_3. Equivalently, the cone has radius $\sqrt{1 - \eta_3^2}$. This situation is depicted in Figure 7.1.

Figure 7.2 shows the vectors uniformly distributed on the 3-sphere that are nearly linear combinations of u_1 and u_2. In this instance, $n = 3$, $d = 2$, and the probability that y_0 lies in the pictured stripe around the 3-sphere is

$$P(\eta_1^2 + \eta_2^2 \geq 1 - \epsilon^2) \geq 1 - \frac{1}{2} \int_0^{1 - \epsilon^2} t^0 (1 - t)^{-\frac{1}{2}} dt.$$

A starting vector lying in this stripe makes an angle of at most $\arcsin \eta_2 < \arcsin \epsilon$ with u_1 or u_2. When $1 - \epsilon^2 = \frac{1}{2}$, the probability that y_0 approximates one eigenvector is at least 0.29, while the probability that it approximates a linear combination of two eigenvectors is 0.71.

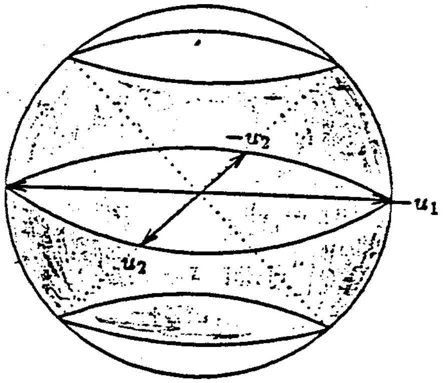

Figure 7.2: Vectors on the Unit 3-Sphere with $\eta_1^2 + \eta_2^2 \geq 1 - \epsilon^2$

7.3 An Analytic Approximation for the Incomplete Beta Function

The probability that a vector $y_0 = (\eta_1, \ldots, \eta_n) = \sum_{i=1}^{n} \eta_i u_i$ approximates a linear combination of the eigenvectors $u_1, \ldots, u_d$ is

$$P(\sum_{i=1}^{d} \eta_i^2 \geq \tau) \geq 1 - \alpha \int_0^{\tau^2} t^{\frac{d}{2}-1}(1-t)^{\frac{n-d}{2}-1} dt, \tag{7.2}$$

where $\alpha = \frac{\Gamma(\frac{n}{2})}{\Gamma(\frac{d}{2})\Gamma(\frac{n-d}{2})}$. The integral in equation (7.2) defines the incomplete beta function

$$\mathcal{I}_{\tau^2}(\frac{d}{2}, \frac{n-d}{2}) = \alpha \int_0^{\tau^2} t^{\frac{d}{2}-1}(1-t)^{\frac{n-d}{2}-1} dt \equiv \alpha \mathcal{I}.$$

Except in the case $n = 3$ or $d = 2$, $\mathcal{I}_{\tau^2}$ generally cannot be evaluated exactly. In determining probability bounds for the condition number of a matrix, Dixon [21] employs an asymptotic estimate for the coefficient α when $d = 1$ and $n \to \infty$ and bounds the integral $\mathcal{I}$ using the mean value theorem for definite integrals

[54] to *estimate* $\mathcal{I}_{\tau^2}(\frac{1}{2},\frac{n-1}{2})$. The mean value theorem for definite integrals states that if the integrand $g(t)$ is continuous over $[\tau_1,\tau_2]$, there exists a number τ_3 such that

$$\int_{\tau_1}^{\tau_2} g(t)dt = g(\tau_3)(\tau_2-\tau_1).$$

Because the integrand of the incomplete beta function attains a maximum value of one on [0,1], the mean value theorem implies that $\mathcal{I}_{\tau^2} \leq 1$.

In this section, an upper bound is established for $\mathcal{I}_{\tau^2}(\frac{d}{2},\frac{n-d}{2})$ for all $d \geq 1$ and $n > d+2$ in two theorems. Theorem 7.3.1 bounds α, and Theorem 7.3.2 bounds $\mathcal{I}$. Following the presentation of the theorems and their proofs, the tightness of the bounds is discussed.

Theorem 7.3.1 *Consider a fixed integer d and any $n > d$. Let*

$$\alpha = \frac{\Gamma(n/2)}{\Gamma(d/2)\Gamma((n-d)/2)}.$$

Then $\alpha > 0$, and

$$\alpha \leq \begin{cases} \sqrt{\frac{(n-2)}{2\pi}}e^{\frac{13}{12}}, & \textit{if } d=1 \textit{ and } n \geq 4; \\ \frac{n}{2}-1, & \textit{if } d=2; \\ \frac{1}{2\sqrt{\pi}}\sqrt{n-2}\left(\frac{n-2}{d-2}\right)^{\frac{d-1}{2}} e^{\frac{d(n-d-1)}{2}-\frac{27}{28}}, & \textit{if } d>2 \textit{ and } n \geq d+3. \end{cases}$$

Proof: The upper limit is determined by use of Stirling's formula [1]

$$\Gamma(\zeta+1) = \sqrt{2\pi}\zeta^{\zeta+\frac{1}{2}}e^{-\zeta+\frac{\theta}{12\zeta}}, \quad \zeta > 0, \quad 0 < \theta < 1. \tag{7.3}$$

Thus,

$$\sqrt{2\pi}\zeta^{\zeta+\frac{1}{2}}e^{-\zeta} \leq \Gamma(\zeta+1) \leq \sqrt{2\pi}\zeta^{\zeta+\frac{1}{2}}e^{-\zeta+\frac{1}{12\zeta}}.$$

Case 1, $d=1$:

Because $\Gamma(\frac{1}{2}) = \sqrt{\pi}$,

$$\alpha = \frac{\Gamma(\frac{n}{2})}{\Gamma(\frac{1}{2})\Gamma(\frac{n-1}{2})} = \frac{1}{\sqrt{\pi}}\frac{\Gamma(\frac{n}{2})}{\Gamma(\frac{n-1}{2})}.$$

Equation (7.3) gives the upper bound

$$\Gamma(\frac{n}{2}) \leq \sqrt{2\pi}\left(\frac{n-2}{2}\right)^{(n-1)/2} e^{-(n/2-1)} e^{\frac{1}{6n-12}}$$

when $n > 2$ and the lower bound

$$\Gamma((n-1)/2) \geq \sqrt{2\pi}(\frac{n-3}{2})^{(n-2)/2} e^{-(n-3)/2}$$

when $n > 3$. Thus,

$$\begin{aligned} \alpha &\leq \frac{1}{\sqrt{\pi}} \frac{((n-2)/2)^{(n-1)/2}}{((n-3)/2)^{(n-2)/2}} e^{-\frac{1}{2}+\frac{1}{6n-12}} \\ &= \frac{1}{\sqrt{2\pi}} \sqrt{n-2}\left(\frac{n-2}{n-3}\right)^{\frac{n-2}{2}} e^{-\frac{1}{2}+\frac{1}{6n-12}}. \end{aligned}$$

The quantity $\phi = \left(\frac{n-2}{n-3}\right)^{\frac{n-2}{2}}$ is bounded as follows

$$\begin{aligned} \ln\phi &= \frac{n-2}{2} \ln\left(\frac{n-2}{n-3}\right) \\ &= \frac{n-2}{2} \ln\left(1+\frac{1}{n-3}\right) \\ &\leq \frac{n-2}{2} \frac{1}{n-3} \\ &= \frac{1}{2}\left(1+\frac{1}{n-3}\right). \end{aligned}$$

The last inequality follows from a Taylor series expansion of $\ln(1+\frac{1}{n-3})$. Therefore, $\phi \leq e^{\frac{1}{2}(\frac{1}{n-3})}$, and

$$\alpha \leq \sqrt{\frac{n-2}{2\pi}} e^{\frac{1}{6n-12}+\frac{1}{2}\left(1+\frac{1}{n-3}\right)}.$$

For $n \geq 4$, $e^{\frac{1}{6n-12}+\frac{1}{2}\left(1+\frac{1}{n-3}\right)} \leq e^{\frac{13}{12}}$.

Case 2, $d = 2$:

Because $\Gamma(\zeta+1) = \zeta\Gamma(\zeta)$ and $\Gamma(1) = 1$,

$$\alpha = \frac{\Gamma(\frac{n}{2})}{\Gamma(1)\Gamma(\frac{n-2}{2})} = \frac{n}{2} - 1.$$

Case 3, $d > 2$:

When $d > 2$ and $n - d > 2$,

$$\begin{aligned} \alpha &= \frac{\Gamma(\frac{n}{2})}{\Gamma(\frac{d}{2})\Gamma(\frac{n-d}{2})} \\ &\leq \frac{1}{\sqrt{2\pi}} \frac{((n-2)/2)^{(n-1)/2}}{((d-2)/2)^{(d-1)/2}((n-d-2)/2)^{(n-d-1)/2}} e^{-1+\frac{1}{6n-12}} \\ &= \frac{1}{2\sqrt{\pi}}\sqrt{n-2}\left(\frac{n-2}{d-2}\right)^{(d-1)/2}\left(\frac{n-2}{n-d-2}\right)^{(n-d-1)/2} e^{-1+\frac{1}{6n-12}}. \end{aligned}$$

By an argument similar to that of Case 1, the quantity $\left(\frac{n-2}{n-d-2}\right)^{\frac{n-d-1}{2}}$ is bounded above by $e^{\frac{d}{2}\left(\frac{n-d-1}{n-d-2}\right)}$, and

$$\begin{aligned} \alpha &\leq \frac{1}{2\sqrt{\pi}}\sqrt{n-2}\left(\frac{n-2}{d-2}\right)^{\frac{d-1}{2}} e^{\frac{1}{6n-12}+\frac{d}{2}\left(\frac{n-d-1}{n-d-2}\right)-1} \\ &\leq \frac{1}{2\sqrt{\pi}}\sqrt{n-2}\left(\frac{n-2}{d-2}\right)^{\frac{d-1}{2}} e^{\frac{d(n-d-1)}{2}-\frac{27}{28}}. \end{aligned}$$

■

Theorem 7.3.2 improves upon the bound from the mean value theorem by taking into account the shape of the integrand. Substitute $s = t^{\frac{d}{2}}$ to obtain

$$\begin{aligned} \mathcal{I} &= \int_0^{\tau^2} t^{\frac{d}{2}-1}(1-t)^{\frac{n-d}{2}-1}dt \qquad (7.4) \\ &= \frac{2}{d}\int_0^{\tau^d}(1-s^{\frac{2}{d}})^{\frac{n-d}{2}-1}ds. \end{aligned}$$

The integrand $f(s) = (1 - s^{\frac{2}{d}})^{\frac{n-d}{2}-1}$ has a single point of inflection at $s = \sigma_I = \left(\frac{d-2}{2-n+2d}\right)^{\frac{d}{2}}$ when σ_I is real and in the interval $(0, 1)$.

Theorem 7.3.2 is based on the following derivation. First consider the case $\tau = 1$ and $\sigma_I \in (0, 1)$. For $s < \sigma_I$, the second derivative $\frac{d^2 f}{ds^2} < 0$, so f is a concave function. For $s > \sigma_I$, $\frac{d^2 f}{ds^2} > 0$, and f is convex. The bound for $\mathcal{I}$ is determined by dividing the area surrounding the integrand into three pieces as depicted in Figure 7.3 for the case $n = 20, d = 1$. The area under the concave portion of f is bounded by the area A_1 of the rectangle of height one and width σ_I. Let δ be a value in $(\sigma_I, 1)$. The area under the convex portion is bounded by the sum of the area A_2 of the triangle with vertices $(\sigma_I, f(\delta))$, $(\sigma_I, f(\sigma_I))$, and $(\delta, f(\delta))$ and the

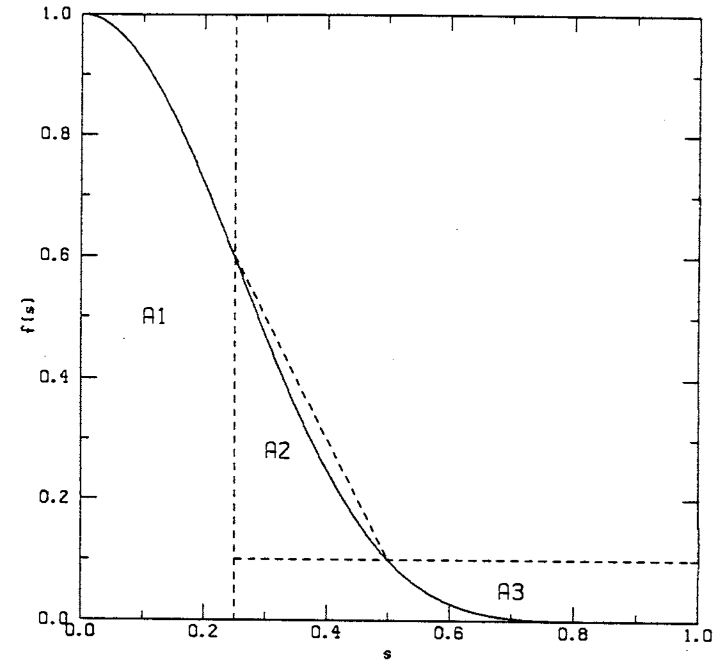

Figure 7.3: The integrand $f(s)$ for $n = 20$ and $d = 1$ *versus* s. The inflection point is $\sigma_I = 0.25$, and $f(\delta) = 0.1$.

area A_3 of the rectangle of height $f(\delta)$ and width $1 - \sigma_I$. The tightness of the bound is dependent on σ_I and on the value of δ selected. If δ is too small, $f(\delta)$ is too large, and the area A_3 is too large; if δ is too large, $f(\delta)$ is too close to zero, and the area A_2 is too large. Regardless of the value of δ, the bound is clearly tighter than that provided by the mean value theorem which bounds the integral by one. As the order n increases, the integrand f falls more steeply toward zero, and the improvement over the mean value theorem increases correspondingly.

When there is no point of inflection (σ_I is non-positive or complex), the integrand $f(s)$ is convex on the interval [0,1]. The area under $f(s)$ is bounded as for the convex portion of the curve when σ_I is in (0,1). This situation is depicted in Figure 7.4 for n=10 and d=3. The area under f is bounded above by the sum of the area A_2 of the shown triangle and the area A_3 of the shown rectangle.

Theorem 7.3.2 defines the bounds for the case $\sigma_I < \delta \leq \tau \leq 1$, which means that area A_3 now represents the area of the rectangle of height $f(\delta)$ and width

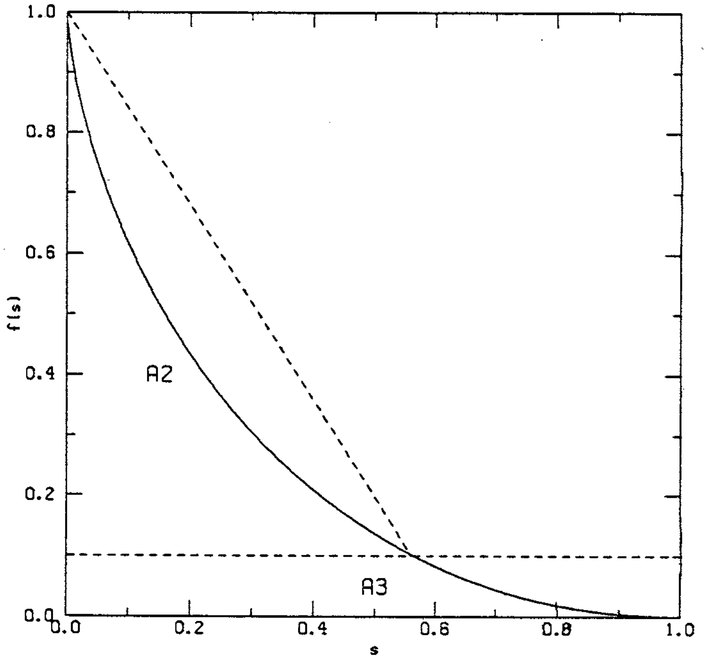

Figure 7.4: The integrand $f(s)$ for $n = 10$ and $d = 3$ *versus* s. There is no inflection point, and $f(\delta) = 0.1$.

$\tau - \sigma_I$. When $\tau \leq \sigma_I$, only the convex portion of the curve is included, and the theorem reduces to the mean value theorem for definite integrals.

Theorem 7.3.2

$$\mathcal{I} \leq \frac{2}{d} \begin{cases} \sigma_I + \frac{1}{2}(\delta - \sigma_I)(f(\sigma_I) - f(\delta)) + (\tau - \sigma_I)f(\delta), & \textit{if } \sigma_I \in (0,1); \\ \frac{1}{2}\delta(1 - f(\delta)) + \tau f(\delta), & \textit{if } \sigma_I \notin (0,1), \end{cases}$$

where $s = \sigma_I$ *is the point of inflection (if it exists) of the integrand* $f(s) = (1 - s^{\frac{2}{d}})^{\frac{n-d}{2}-1}$ *in the interval (0,1), and* $\sigma_I < \delta \leq \tau \leq 1$.

Proof: Let $f(s) = (1 - s^{\frac{2}{d}})^{\frac{n-d}{2}-1}$ and $\sigma = \max(0, \sigma_I) < 1$. If σ_I is complex, let $\sigma = 0$. The above bound is then derived by writing

$$\mathcal{I} = \int_0^\sigma f(s)ds + \int_\sigma^\delta f(s)ds + \int_\delta^\tau f(s)ds.$$

Note that $f(s)$ is decreasing for s in [0,1], so

$$\int_0^\sigma f(s)ds \leq \sigma$$

by the mean value theorem for definite integrals [54].

For $s > \sigma$, f is convex. Therefore, all points on the line l drawn between the points $(\sigma, f(\sigma))$ and $(\delta, f(\delta))$ are no less than any value of $f(s)$ for $s \in [\sigma, \delta]$, and the integral is bounded by the trapezoidal rule

$$\int_\sigma^\delta f(s)ds \leq \frac{1}{2}(\delta - \sigma)(f(\sigma) - f(\delta)) + (\delta - \sigma)f(\delta),$$

where the righthand side is the area under the line l.

By the mean value theorem, the remaining portion of the integral is bounded above by

$$\int_\delta^\tau f(s)ds \leq f(\delta)(\tau - \delta).$$

■

We first assess the accuracy of the bounds in Theorems 7.3.1 and 7.3.2 for the case $d = 1$ and $\tau = 1$ by comparing the theoretical bounds to the results of Gauss-Legendre quadrature. The number of quadrature nodes is chosen so that increasing the number of nodes does not alter the eight most significant digits of the result. Fifty to one hundred nodes suffice for the matrix orders tested. Because the integrand $f(s)$ falls rapidly toward zero, the results are not strongly dependent on τ unless τ is very small.

The coefficient α bounded in Theorem 7.3.1 is just the reciprocal of the beta function $\beta(\frac{d}{2}, \frac{n-d}{2})$ and so is equal to

$$\left[\frac{2}{d}\int_0^1 (1 - s^{\frac{2}{d}})^{\frac{n-d-2}{2}}\right]^{-1}. \tag{7.5}$$

Table 7.1 lists the bounds for α from Theorem 7.3.1 when $d = 1$ and $n = 100$, 1000, and 10000 together with the reciprocal of the beta function in equation (7.5) computed by Gauss-Legendre integration. The last column shows the ratio of the theoretical and computed bounds. The ratios show that the theorem's bounds are very tight for the case $d = 1$.

The accuracy of Theorem 7.3.2 is demonstrated in Tables 7.2 and 7.3. The second and third columns of Table 7.2 show, respectively, the bound determined by Theorem 7.3.2 and the value of the integral $\mathcal{I}$ when $\tau = 1$ computed by Gauss-Legendre quadrature for $\delta = .01$ and n ranging from 100 to 10000. The last column shows the ratios of the bounds given by the mean value theorem (1.0 for all n) and the computed integrals. For all tests, the point of inflection σ_I lies in the s-interval (0,1). The fourth column shows the ratios of the bounds and computed integrals. As n grows, the integrand $f(s)$ falls with increasing steepness so that the tightness of the bound in Theorem 7.3.2 decreases. For the same reason, however, the improvement over the mean value theorem increases significantly.

Table 7.3 shows the same results when $\delta = 10^{-8}$. Although the bounds are not as tight for small values of n as when $\delta = .01$, they still provide a marked improvement over those of the mean value theorem. For the range of orders shown, $\delta = .01$ provides roughly the best accuracy, though the change in accuracy with δ is not very large.

n	Theorem 7.3.1 Bound	Computed Value	$\frac{\text{Theorem}}{\text{Computed}}$
100	3.96	3.96	1.0
1000	12.61	12.61	1.0
10000	39.89	39.89	1.0

Table 7.1: Comparison of theoretical bounds for α from Theorem 7.3.1 with computed values for $d = 1$.

n	Theorem 7.3.2 Bound	Computed Value	$\frac{\text{Theorem}}{\text{Computed}}$	$\frac{\text{MVT}}{\text{Computed}}$
100	0.3394	0.2525	1.3	4.0
200	0.2459	0.1779	1.4	5.6
500	0.1628	0.1123	1.4	8.9
1000	0.1210	0.0916	1.5	12.6
5000	0.0652	0.0793	1.8	28.2
10000	0.0519	0.0354	2.0	39.8

Table 7.2: Comparison of theoretical bounds from Theorem 7.3.2 and from the mean value theorem with computed values for $d = 1$ and $\delta = 1.d - 2$.

When $d = 2$, the integral becomes $\mathcal{I} = \int_0^{\tau^2} (1-s)^{\frac{n-2}{2}} ds$, and it can be integrated exactly. When $d > 2$, the integrand $f(s)$ is very steep and difficult to integrate numerically. For example, when $d = 3$ and $n = 1000$, $f(s) < 10^{-10}$ for all $s > .01$. Approximate computed values for $\mathcal{I}$ are listed in Table 7.4 for $n = 100$ and $n = 1000$ when $d = 3$ and $\delta = .01$ along with the bounds from Theorem 7.3.2 and the mean value theorem. The theorem's bounds are considerably less tight than when $d = 1$ but nevertheless provide a large improvement over those of the mean value theorem. As n or d increases and the integrand steepens, the advantage of Theorem 7.3.2 over the mean value theorem increases even though the tightness of the theorem's bounds decreases.

7.4 The Quality of the Starting Vectors

In Chapter 6, it was demonstrated that random vectors make good starting vectors for inverse iteration. Experimentally, a random starting vector is typically not orthogonal to the eigenvector being computed, and the specific random vectors used turned out to be linearly independent. In this section, the analytic

n	Theorem 7.3.2 Bound	Computed Value	$\frac{\text{Theorem}}{\text{Computed}}$	$\frac{\text{MVT}}{\text{Computed}}$
100	0.4838	0.2525	1.9	4.0
1000	0.1597	0.0793	2.0	12.6
10000	0.0507	0.0251	2.0	39.8

Table 7.3: Comparison of theoretical bounds from Theorem 7.3.2 and from the mean value theorem with computed values for $d = 1$ and $\delta = 1.d - 8$.

n	Theorem 7.3.2 Bound	Computed Value	$\frac{\text{Theorem}}{\text{Computed}}$	$\frac{\text{MVT}}{\text{Computed}}$
100	.1	.08	1.25	12.5
1000	0.005	0.0002	25	5000

Table 7.4: Comparison of theoretical bounds from Theorem 7.3.2 and from the mean value theorem with computed values for $d = 3$ and $\delta = .01$.

definition of a good starting vector given in equation (7.1) is used to explain this observation statistically and to establish the number of times a starting vector can be reused when eigenvalues are well-spaced. In this analysis, it is assumed that a vector $x_0 = (\xi_1, \ldots, \xi_n)^T$ has independent, normally distributed random components. The starting vector is formed by normalizing x_0 to form $y = \sum_{i=1}^{n} \eta_i u_i$. For rapid computation of $\hat{u}_i$ using $\lambda = \hat{\lambda}_i$, it is essential that y have a large enough component in the u_i direction. The probability that the coefficient η_i is bounded below by $\sqrt{1-\epsilon^2}$ is restated as Theorem 7.4.1:

Theorem 7.4.1 *Let* $x_0 = (\xi_1, \ldots, \xi_n)^T$*, have independent random components* ξ_i *each with a normal (0,1) distribution and let* $y_0 = x_0 / \| x_0 \|_2 = (\eta_1, \ldots, \eta_n)^T$*. Given* $0 \le \epsilon \le 1$*, the probability that* $\eta_i^2 \ge 1 - \epsilon^2$ *is*

$$P(\eta_i^2 \ge 1 - \epsilon^2) \ge 1 - \alpha \int_0^{1-\epsilon^2} t^{-\frac{1}{2}} (1-t)^{\frac{n-3}{2}} dt. \qquad (7.6)$$

Table 7.5 gives these probabilities for matrix orders 100, 1000, and 10000 for a range of $1 - \epsilon^2$ values. The integral in equation (7.6) was computed by Gauss-Legendre quadrature using 100 nodes. The probabilities establish that a random starting vector is expected to have a component of magnitude sufficient for fast convergence of inverse iteration in any eigenvector direction. For $n \le 10000$, the

$\sqrt{1-\epsilon^2}$	for $n = 100$ $P(\lvert\eta_i\rvert \geq \tau = \sqrt{1-\epsilon^2})$	for $n = 1000$ $P(\lvert\eta_i\rvert \geq \tau = \sqrt{1-\epsilon^2})$	for $n = 10000$ $P(\lvert\eta_i\rvert \geq \tau = \sqrt{1-\epsilon^2})$
$< 10^{-4}$	1.00 (16)	1.00 (16)	1.00 (16)
10^{-3}	0.99	0.97	0.90
10^{-2}	0.92	0.76	0.34
10^{-1}	0.34	0 (2)	0 (16)
$> .7$	0 (16)	0 (16)	0 (16)

Table 7.5: Lower bounds on probability that $|\eta_i| \geq \tau$. Numbers in parentheses equal the number of zero decimal places.

probability that any one component is at least 10^{-4} is 1 ± 10^{-16}. The probability bounds are still at least 0.9 for $\sqrt{1-\epsilon^2} = .001$ for all three orders. For a given tolerance $1 - \epsilon^2$, the probability bounds decrease as n increases: as the number of components in a vector increases, the probability that any one component is large decreases. The steep drops as $\sqrt{1-\epsilon^2}$ increases correspond to large changes in the integrand as described in Section 7.3. Note that the unit norm of the starting vector guarantees that not all components can be very small compared to one.

The sets of n randomly generated starting vectors used in the experiments in Chapter 6 were also linearly independent. The following theorem supports this observation statistically. Namely, a set of n vectors $\{x_1, \ldots, x_n\}$ is numerically linearly dependent, if for all sets of nonzero coefficients $\alpha_1, \alpha_2, \ldots, \alpha_n$ such that $\sum_{i=1}^n \alpha_i^2 = 1$, the norm $\| z \|_2 = \| \sum_{i=1}^n \alpha_i x_i \|_2 \leq \delta$. Note that exact linear independence would require $\| z \|_2 = 0$ and that $P(\| z \|_2 = 0) = 0$.

Theorem 7.4.2 (Linear Independence of Starting Vectors) *Let the column vectors $x_i = (\xi_{1i}, \xi_{2i}, \ldots, \xi_{ni})^T$, $1 \leq i \leq n$, have independent random components each with a normal (0,1) distribution, and let $\alpha_1, \alpha_2, \ldots, \alpha_n$ be real numbers such that $\sum_{i=1}^n \alpha_i^2 = 1$. Let $z = \sum_{i=1}^n \alpha_i x_i = (\zeta_1, \zeta_2, \ldots, \zeta_n)^T$. For fixed $\epsilon > 0$, the probability that $\| z \|_2 \leq \epsilon$ satisfies*

$$P(\| z \|_2 \leq \epsilon) \leq 2n(\epsilon/\sqrt{2\pi}).$$

$P(\| z \|_2 \leq \epsilon)$ is then the probability that the vectors $x_1, \ldots, x_n$ are linearly dependent to within a tolerance ϵ.

Proof: Each element ξ_{ij} has a normal (0,1) distribution, and the sum $\sum_{i=1}^{n} \alpha_i^2 = 1$. By Lemma 7.8.3, $\zeta_i = \sum_{i=1}^{n} \alpha_i \xi_{ij}$ has a normal (0,1) distribution and probability density function $f(t) = \frac{1}{\sqrt{2\pi}} e^{-t^2/2}$. According to equation (7.10), the probability

$$\begin{aligned} P(\| z \|_\infty \leq \epsilon) &= P(|\zeta_i| \leq \epsilon, i = 1, \ldots, n) \\ &\leq \sum_{i=1}^{n} P(|\zeta_i| \leq \epsilon) \\ &= nP(|\zeta_1| \leq \epsilon). \end{aligned}$$

Dividing both sides by n gives

$$\begin{aligned} \frac{1}{n} P(\| z \|_\infty \leq \epsilon) &\leq F(\epsilon) - F(-\epsilon) \\ &= \int_{-\infty}^{\epsilon} f(x)dx - \int_{-\infty}^{-\epsilon} f(x)dx \\ &= 2\int_{0}^{\epsilon} f(x)dx \\ &\leq \frac{2\epsilon}{\sqrt{2\pi}}. \end{aligned}$$

Because $\| z \|_\infty \leq \| z \|_2$, $P(\| z \|_\infty \leq \epsilon) \geq P(\| z \|_2 \leq \epsilon)$, and

$$P(\| z \|_2 \leq \epsilon) \leq 2n(\epsilon/\sqrt{2\pi}).$$

■

When the eigenvalues of the matrix are well-separated, linear independence of the starting vectors seems less essential. In this case, efficiency is improved by reusing random starting vectors for different eigenvector computations. The same starting vector can be used for eigenvectors 1 through d as long as $\eta_i^2 \geq 1 - \epsilon^2$ for a given $1 - \epsilon^2$ and $i = 1, \ldots, d$. The following theorem gives an expression for the number of times d a starting vector can be reused with probability ρ.

Theorem 7.4.3 (Reuse of Starting Vectors) *Let* $x_0 = (\xi_1, \ldots, \xi_n)^T$, $n \geq 2$, *have independent random components with normal (0,1) distributions, and* $y_0 = x_0/\| x_0 \|_2 = (\eta_1, \ldots, \eta_n)^T$. *Then* $\eta_i^2 \geq 1 - \epsilon^2$ *for* $i = 1, \ldots, d$ *with probability at least* ρ *if* $d \leq \lfloor \frac{1-\rho}{\alpha I} \rfloor$.

ρ	$\tau = \sqrt{1-\epsilon^2}$	for $n = 100$ d	for $n = 1000$ d	for $n = 10000$ d
0.5	$< 10^{-4}$	100	1000	10000
	10^{-3}	50	16	5
	10^{-2}	6	2	< 1
	10^{-1}	< 1	< 1	< 1
0.9	$< 10^{-4}$	100	1000	10000
	10^{-3}	10	3	1
	10^{-2}	1	< 1	< 1
	10^{-1}	< 1	< 1	< 1
0.99	$< 10^{-4}$	100	1000	10000
	10^{-3}	1	< 1	< 1
	10^{-2}	< 1	< 1	< 1
	10^{-1}	< 1	< 1	< 1

Table 7.6: The number of times d a starting vector can be used with probability ρ that $|\eta_i| \geq \tau$, $i = 1, \ldots, d$.

Proof: According to equation (7.9),

$$\begin{aligned} P(\eta_i^2 \geq 1-\epsilon^2, i = 1, \ldots, d) &= 1 - P(\eta_i^2 < (1-\epsilon^2), i = 1, \ldots, d) \\ &\geq 1 - \sum_{i=1}^{d} P(\eta_i^2 \leq (1-\epsilon^2)) \\ &= 1 - d\alpha\mathcal{I}. \end{aligned}$$

Setting $\rho = 1 - d\alpha\mathcal{I}$ and rearranging terms gives the expression $d = \lfloor \frac{1-\rho}{\alpha\mathcal{I}} \rfloor$. ■

Table 7.6 shows values of s for several choices of $\sqrt{1-\epsilon^2}$ when $n = 100$, 1000, 10000. As matrix order increases with ρ and $\sqrt{1-\epsilon^2}$ fixed, the number of times a vector can be reused decreases. This echoes the trend observed in Table 7.5: a long vector of norm one is less likely to have large components and so is less acceptable for reuse. As a component with norm 10^{-4} appears sufficient for rapid convergence, the same starting vector may be reused for all eigenvectors with probability at least 0.99 for matrix orders 100, 1000, and 10000.

7.5 Application of Statistics to Iterates without Reorthogonalization

Statistical analysis can be applied to the results of inverse iteration as well. Unfortunately, the analysis is considerably less informative than that for the starting vectors. This section discusses inverse iteration without reorthogonalization. The kth iterate in the computation of the eigenvector corresponding to the computed eigenvalue λ is defined as follows:

Algorithm 7.5.1 (*k* Inverse Iterations to Compute One Eigenvector) *1. Generate a random vector $x_0 = (\xi_1, \ldots, \xi_n)^T$ with independent random components each having a normal (0,1) distribution.*

2. Normalize x_0 to produce the starting vector

$$y = \frac{x_0}{\| x_0 \|_2} = (\eta_1, \ldots, \eta_n)^T = \sum_{i=1}^{n} \eta_i u_i.$$

3. Perform k inverse iterations to produce the unnormalized iterate

$$z = (T - \lambda)^{-k} y = (\zeta_1, \ldots, \zeta_n)^T = \sum_{i=1}^{n} \zeta_i u_i.$$

The probability that the iterate z approximates a single eigenvector u_i of the matrix T is given in Theorem 7.5.1.

Theorem 7.5.1 *Let $x_0 = (\xi_1, \ldots, \xi_n)^T$, $n \geq 2$, have independent random components ξ_i each with a normal (0,1) distribution, and let $y = x_0 / \| x_0 \|_2 = (\eta_1, \ldots, \eta_n)^T$. Let the kth unnormalized, unorthogonalized iterate be defined by*

$$z = (T - \lambda)^{-k} y = (\zeta_1, \ldots, \zeta_n)^T.$$

Given $0 \leq \epsilon \leq 1$, the probability that $\zeta_i^2 \geq 1 - \epsilon^2$ for some i is

$$P(\zeta_i^2 \geq 1 - \epsilon^2) \geq 1 - \alpha \int_0^{(\lambda_i - \lambda)^{2k}(1 - \epsilon^2)} t^{-\frac{1}{2}} (1 - t)^{\frac{n-3}{2}} dt. \tag{7.7}$$

If $(1 - \epsilon^2)(\lambda_i - \lambda)^{2k} \geq 1$, then $P(\zeta_i^2 \geq 1 - \epsilon^2) \geq 0$.

Proof: The jth component of iterate z is

$$\zeta_j^2 = \frac{\eta_j^2}{(\lambda_j - \lambda)^{2k}} = \frac{\xi_j^2}{(\lambda_j - \lambda)^{2k}} / \sum_{i=1}^{n} \xi_j^2.$$

Thus, the probability that $\zeta_j^2 \geq 1 - \epsilon^2$ is the probability that

$$\eta_j^2 = \frac{\xi_j^2}{\sum_{i=1}^{n} \xi_j^2} \geq (1 - \epsilon^2)(\lambda_j - \lambda)^{2k}$$

given by equation (7.6) in Theorem 7.4.1. Invoking Theorem 7.4.1 completes the proof. ■

By the mean value theorem, $\mathcal{I}$ is bounded above by $(1 - \epsilon^2)(\lambda_j - \lambda)^{2k}$. Hence, if λ is a close approximation to λ_j, $\mathcal{I}$ is small, and the probability that z approximates u_j is close to one. If $\lambda_j - \lambda = \lambda_{j+1} - \lambda$, z approximates u_j and u_{j+1} with equal probability.

A potentially more interesting statistic concerns the behavior of the iterates when λ_j and λ_{j+1} are close but not equal. The results, however, are difficult to interpret in this case. One of the two eigenvectors u_j and u_{j+1} will be better approximated than the other only if $s_j = (1 - \epsilon^2)(\lambda_j - \lambda)^{2k}$ and $s_{j+1} = (1 - \epsilon^2)(\lambda_{j+1} - \lambda)^{2k}$ lie where the integrand $f(t) = t^{-\frac{1}{2}}(1 - t)^{\frac{n-3}{2}}$ has a large derivative. Only in this case will the change of upper limit from s_j to s_{j+1} significantly change the value of the integral. Thus, although the effects of additional iterations on the accuracy of the solution can be quantified in terms of the integral $\mathcal{I}$, a qualitative interpretation of the result is difficult in general.

Just as the statistical analysis falls short in determining the preferred number of iterations, it fails on other possible stopping criteria. Determining the expected iterate norm $\| z \|_2$ or the residual $\| Tz - \lambda z \|_2$ of a computed eigenpair involves sums of the form

$$S = \sum_{j=1}^{n} \alpha_j^2 \zeta_j^2 = \sum_{j=1}^{n} \frac{\alpha_j^2 \xi_j^2}{(\lambda_j - \lambda)^{2k}} / \sum_{j=1}^{n} \eta_j^2,$$

where, respectively, $\alpha_j = 1$ for all j or $\alpha_j = \lambda_j - \lambda$. These sums also arise in the analysis of the normalized iterate $\frac{z}{\|z\|_2}$.

The distribution of the sum S depends on the distribution of both its numerator and denominator. While the distribution of the latter is $\Gamma(\frac{n}{2}, 2)$, that of the former generally cannot be expressed in closed form. Moreover, even the simplest approximation of the distribution of the numerator is unwieldy [12, 48]. Analysis of the orthogonality of two computed iterates runs into similar problems. Thus, a probabilistic analysis of the stopping criterion or reorthogonalization criterion appears inaccessible.

7.6 The Quality of Iterates After Reorthogonalization

Extending the statistical analysis to the iterates after reorthogonalization requires deriving a probability density function for their components. This is done by expressing the components as functions of random variates having known distribution. A single inverse iteration using random starting vectors proceeds as follows. The orthogonal basis $U = (u_1 \dots u_n)$ of eigenvectors of $T = U\Lambda U^T$ is used for all vectors in the following analysis.

Algorithm 7.6.1 (One Inverse Iteration to Compute All Eigenvectors)

1. *Generate random vectors* $x_i = (\xi_{1i}, \dots, \xi_{ni})^T$, $i = 1, \dots, n$, *where the components* x_{mi} *are independent random variables with normal (0,1) distributions.*

2. *Normalize the random vectors to produce the unit starting vectors* $y_i = \frac{x_i}{\|x_i\|_2} = (\eta_{1i}, \dots, \eta_{ni})^T$, $i = 1, \dots, n$.

3. *Form*

$$z_i = (T - \hat{\lambda}_i)^{-1} y_i, \quad i = 1, \dots, n.$$

4. *Orthogonalize the unnormalized iterates* $Z = (z_1, \dots, z_n) = QR$ *so that* $Q = ZR^{-1}$ *is orthogonal and* R *is upper triangular.*

Denote the element $e_i^T R^{-1} e_j$ by ρ_{ij}, then the ith orthogonalized iterate q_i may be written

$$\begin{aligned} q_i &= \sum_{k=1}^{i} \rho_{ki} z_k \\ &= \sum_{k=1}^{i} \rho_{ki} (T - \hat{\lambda}_k)^{-1} y_k \\ &= \sum_{k=1}^{i} \rho_{ki} (T - \hat{\lambda}_k)^{-1} \sum_{l=1}^{n} \eta_{lk} u_l. \end{aligned}$$

The jth component of orthogonalized iterate q_i is given by

$$u_j^T q_i = \sum_{k=1}^{i} \frac{\rho_{ki}}{\lambda_j - \hat{\lambda}_k} \eta_{jk}.$$

This can be written in terms of the normally distributed components of the random vectors x_m:

$$u_j^T q_i = \sum_{k=1}^{i} \frac{\rho_{ki}}{\lambda_j - \hat{\lambda}_k} \xi_{jk} / \left(\sum_{l=1}^{n} \xi_{lk}^2 \right)^{\frac{1}{2}}.$$

The numerator of this term is a linear combination of normal (0,1) distributed variables and so has a normal $(0, \sum_{k=1}^{i} \frac{\rho_{ki}}{\lambda_j - \hat{\lambda}_k})$ distribution. Note that the elements ρ_{ki} are themselves derived from random quantities. The square of the denominator is the sum of squares of normally distributed elements and so has a $\Gamma(\frac{n}{2}, 2)$ distribution. The probability density function (pdf) of $u_j^T q_i$ is given in Lemma 7.6.1.

Lemma 7.6.1 *The probability density function of $u_j^T q_i$ is*

$$\frac{1}{\sqrt{\psi}} \frac{\gamma(\frac{n+1}{2})}{\gamma(\frac{1}{2})\gamma(\frac{n}{2})} \left(\frac{z^2}{\psi} + 1 \right)^{-(\frac{n+1}{2})},$$

where $\psi = \sum_{k=1}^{i} \frac{\rho_{ki}}{\lambda_j - \hat{\lambda}_k}$.

Proof: The pdf of $u_j^T q_i$ is the pdf of $\frac{x}{\sqrt{y}}$ where

$$\begin{aligned} x &= \sum_{k=1}^{i} \frac{\rho_{ki}}{\lambda_j - \hat{\lambda}_k} \xi_{jk} \\ y &= \sum_{l=1}^{n} \xi_{lk}^2. \end{aligned}$$

To derive this pdf, let

$$\begin{aligned} z &= \frac{x}{\sqrt{y}} \\ w &= y. \end{aligned}$$

Then the Jacobian of the inverse transformation is

$$J = \begin{vmatrix} \frac{\partial x}{\partial z} & \frac{\partial x}{\partial w} \\ \frac{\partial y}{\partial z} & \frac{\partial y}{\partial w} \end{vmatrix} = \begin{vmatrix} \sqrt{w} & \frac{z}{2\sqrt{w}} \\ 0 & 1 \end{vmatrix} = \sqrt{w}.$$

Thus, (z, w) has the joint density

$$\left[\sqrt{2\pi\psi}\gamma\frac{n}{2}2^{n/2}\right]^{-1} w^{\frac{n-1}{2}} e^{-\frac{w}{2}\left(\frac{z^2}{\psi}+1\right)} \quad = \quad \kappa w^{\frac{n-1}{2}} e^{-\frac{w}{2}\left(\frac{z^2}{\psi}+1\right)}$$

We obtain the marginal density for z by integrating with respect to w.

$$\begin{aligned} f &= \kappa \int_0^\infty w^{\frac{n-1}{2}} e^{-\frac{w}{2}\left(\frac{z^2}{\psi}+1\right)} \\ &= \frac{1}{\sqrt{\psi}} \frac{\gamma(\frac{n+1}{2})}{\gamma(\frac{1}{2})\gamma(\frac{n}{2})} \left(\frac{z^2}{\psi}+1\right)^{-\left(\frac{n+1}{2}\right)}. \end{aligned}$$

■

For this result to hold, R^{-1} must exist, and ψ must be positive. The first assumption holds whenever the columns of Z are linearly independent. Because the Modified Gram-Schmidt procedure fails when columns of Z are linearly dependent, the existence of R^{-1} is established in the course of the iteration. The second assumption, however, requires determining the differences $\lambda_j - \hat{\lambda}_k$ for $j, k = 1, \ldots, n$, where λ_j for $j = 1, \ldots, n$ are the *exact* eigenvalues of the matrix T. It also requires explicit computation of the inverse R^{-1}. Furthermore, there is no reason to expect that ψ is positive in general. Thus, the above probability density function cannot be determined without actually solving the problem and performing additional and expensive computation, and the necessary assumptions may not hold. For these reasons, the statistical analysis does not readily apply to the orthogonalized iterates.

	for $n = 100$	for $n = 1000$	for $n = 10000$
$\sqrt{1-\epsilon^2}$	$P(\lvert\eta_i\rvert \geq \tau = \sqrt{1-\epsilon^2})$	$P(\lvert\eta_i\rvert \geq \tau = \sqrt{1-\epsilon^2})$	$P(\lvert\eta_i\rvert \geq \tau = \sqrt{1-\epsilon^2})$
$\leq 10^{-6}$	1.00 (16)	1.00 (16)	1.00 (16)
10^{-5}	1.00 (16)	0.99	0.90
10^{-4}	0.99	0.36	0.34
10^{-3}	0.92	0.33	0 (16)
10^{-2}	0.34	0 (2)	0 (16)
$\geq 10^{-1}$	0 (16)	0 (16)	0 (16)

Table 7.7: Lower Bounds on probability that $|\eta_i| \geq \tau$. Numbers in parentheses equal the number of zero decimal places.

7.7 Practical Considerations

The preceding statistical analysis qualitatively confirms the experimental results of Chapter 6. This analysis, however, assumes generation of independent, normally distributed variables, while the experiments were performed using uniformly distributed variables in [-1,1] having some degree of dependence. Thus, the experimental starting vectors of length n are uniformly distributed on a hypercube of dimension n and height 2 centered at the origin rather than on the unit n-sphere as assumed in the analysis. Normally distributed pseudorandom numbers can be generated at a somewhat higher cost than the uniform ones [20], but this section shows that uniform ones are acceptable substitutes.

The error in applying the analysis to uniformly distributed starting vectors may be estimated by circumscribing an n-sphere of radius $\sqrt{n}$ about the hypercube. A vector $y = (\eta_1, \ldots, \eta_n)^T = \sum_{i=1}^n \eta_i u_i$ on the sphere is a good approximation to a multiple $\sqrt{n}u_i$ of the eigenvector u_i if $|\eta_i| \geq \sqrt{n(1-\epsilon^2)}$. Following the derivation in Section 7.2, the probability of this occurrence is

$$P(\eta_i^2 \geq n(1-\epsilon^2)) = 1 - \alpha \int_0^{n(1-\epsilon^2)} t^{-\frac{1}{2}}(1-t)^{\frac{n-3}{2}} dt.$$

Lower bounds for this probability computed by Gauss-Legendre quadrature are given in Table 7.7.

The probability of a large component η_i is not as high as in the normally distributed case, but components with magnitude 10^{-6} can be expected with near certainty. The experiments in Chapter 6, show that this magnitude is sufficient for convergence. This result does not hinge on the linear dependence of components.

In addition to having large enough components, it is essential that the starting vectors be linearly independent. The possibility of linear independence is dependent on the random number generator used. For the experiments in Chapter 6, an $n \times n$ matrix of random numbers was generated using the routine RAND available from NETLIB; the columns of this matrix formed the n starting vectors. The linear independence of the columns was demonstrated in Chapter 6.

7.8 Appendix: Statistical Basics

The proofs of the theorems presented in this chapter depend on a few statistical fundamentals reviewed in this appendix.

7.8.1 Definitions

A real random variable ξ is characterized by its *distribution function* which is defined as the probability that $\xi \leq \omega$ for some real ω. The distribution function is then denoted $F(\omega) = P(\xi \leq \omega)$. If $F(\omega)$ is absolutely continuous and differentiable, it may be expressed as the integral $F(\omega) = \int_{-\infty}^{\omega} f(x)\, dx$. Any function f for which this integral exists is termed a *probability density function* (pdf) of ξ. For most values of ω, $\frac{d}{d\omega}F(\omega) = f(\omega)$.

The quantity $E(\xi^{\alpha}) \equiv \int_{-\infty}^{\xi} x^{\alpha} f(x)\, dx$ is the *α-th moment* of ξ about zero. The first moment $E(\xi)$ is the *mean* of the distribution; $E((\xi - E(\xi))^2) = E(\xi^2) - E^2(\xi)$ is its variance. The integral $\phi(t) = E(e^{it\xi}) = \int_{-\infty}^{\xi} e^{itx} f(x)\, dx$ is the Fourier transform of $f(x)$ and is known as the *characteristic function* of ξ. A distribution is uniquely specified by $F(\omega)$ or by $\phi(t)$.

These univariate definitions can be extended for sets of random variables. Specifically, the real random vector $(\xi_1, \xi_2, \ldots, \xi_n)$ has the distribution function

$$F(\omega_1, \omega_2, \ldots, \omega_n) = P(\xi_1 \leq \omega_1, \xi_2 \leq \omega_2, \ldots, \xi_n \leq \omega_n).$$

If F is absolutely continuous and the integral

$$F(\omega_1, \omega_2, \ldots, \omega_n) = \int_{-\infty}^{\omega_1} \cdots \int_{-\infty}^{\omega_n} f(x_1, \ldots, x_n)\, dx_1 \cdots dx_n$$

exists, f is a *joint probability density function* of $\xi_1, \xi_2, \ldots, \xi_n$. Random variates $\xi_1, \ldots, \xi_n$ are independent if and only if $F(\omega_1, \ldots, \omega_n) = F(\omega_1) \ldots F(\omega_n)$.

The following two probability relations are also used in the proofs in this chapter:

$$\begin{aligned} P(\xi \geq \omega) &= 1 - P(\xi < \omega) && (7.8) \\ &\geq 1 - P(\xi \leq \omega) && (7.9) \end{aligned}$$

$$\begin{aligned} P(\xi_1 \geq \omega_1 \text{ and } \xi_2 \geq \omega_2) &= P(\xi_1 \geq \omega_1) + P(\xi_2 \geq \omega_2) - P(\xi_1 \geq \omega_1 \text{ or } \xi_2 \geq \omega_2) \\ &\leq P(\xi_1 \geq \omega_1) + P(\xi_2 \geq \omega_2). && (7.10) \end{aligned}$$

Three relevant univariate distributions are the normal (μ, σ^2) distribution with mean μ and variance σ^2, the gamma distribution $\Gamma(\alpha_1, \alpha_2)$, and the beta distribution $B(\alpha_1, \alpha_2)$. $\Gamma(\frac{1}{2}, 2)$ is the chi-square distribution χ^2, and $\Gamma(\frac{\alpha_1}{2}, 2)$ is the chi-square distribution with α_1 degrees of freedom $\chi^2_{\alpha_1}$. Table 7.8 summarizes some properties of these distributions. The gamma function is defined by

$$\gamma(\alpha_1) \equiv \int_0^\infty t^{\alpha_1 - 1} e^{-t} dt$$

for $\alpha_1 > 0$ and the beta function by

$$\beta(\alpha_1, \alpha_2) \equiv \int_0^1 t^{\alpha_1 - 1}(1-t)^{\alpha_2 - 1} dt = \frac{\gamma(\alpha_1)\gamma(\alpha_2)}{\gamma(\alpha_1 + \alpha_2)}$$

for $\alpha_1, \alpha_2 > 0$.

7.8.2 Lemmas

Versions of the lemmas given in this section and their proofs may be found, for example, in [20, 70].

distribution	pdf	characteristic function
Normal(μ, σ^2)	$\frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/(2\sigma^2)}$	$e^{i\mu t - \sigma^2 t^2/2}$
$\Gamma(\alpha_1, \alpha_2)$	$\frac{1}{\gamma(\alpha_1){\alpha_2}^{\alpha_1}} x^{\alpha_1 - 1} e^{-x/\alpha_2}$, $x > 0$	$(1 - it\alpha_2)^{-\alpha_1}$
$\mathrm{B}(\alpha_1, \alpha_2)$	$\frac{\gamma(\alpha_1+\alpha_2)}{\gamma(\alpha_1)\gamma(\alpha_2)} x^{\alpha_1 - 1}(1-x)^{\alpha_2 - 1}$, $0 \leq x \leq 1$	not needed

Table 7.8: Properties of some distributions.

Lemma 7.8.1 *If ζ is random variable with a normal (0,1) distribution, ζ^2 has a $\Gamma(\frac{1}{2}, 2)$ distribution.*

Lemma 7.8.2 *If $\eta_1, \ldots, \eta_n$ are independent random variables with respective characteristic functions $\phi_j(t)$, $j = 1, 2, \ldots, n$, then the characteristic function of $\sigma = \sum_{i=1}^{n} \gamma_i \eta_i$ is $\phi(t) = \prod_{j=1}^{n} \phi_j(\gamma_j t)$ for real numbers $\gamma_j, j = 1, \ldots, n$.*

Lemma 7.8.3 *If $\eta_1, \eta_2, \ldots, \eta_n$ are independent random variables each having a normal $(0,1)$ distribution, then $\sigma = \sum_{j=1}^{n} \gamma_j \eta_j$ has a normal $(0, \sum_{j=1}^{n} \gamma_j)$ distribution.*

Lemma 7.8.4 *If $\eta_1, \eta_2, \ldots, \eta_n$ are independent random variables each having a $\Gamma(\alpha_1, \alpha_2)$ distribution, then $\sigma = \sum_{i=1}^{n} \eta_i$ has a $\Gamma(\alpha_1 n, \alpha_2)$ distribution.*

Lemma 7.8.5 *If η_1 and η_2 are independent random variables having respective distributions $\Gamma(\alpha_1, \phi)$ and $\Gamma(\alpha_2, \phi)$ then $\eta_1/(\eta_1 + \eta_2)$ has the distribution $B(\alpha_1, \alpha_2)$.*

Chapter 8

The Bidiagonal SVD

The singular value decomposition (SVD) of a real $n \times n$ bidiagonal matrix B can be written

$$B = Y \Sigma X^T,$$

where Y and X are both orthogonal matrices and Σ is a diagonal matrix with non-negative diagonal elements. The columns of Y and X are, respectively, the left and right singular vectors of B; the diagonal elements of Σ are its singular values. This chapter compares ways of computing the SVD of a bidiagonal matrix serially and on a distributed-memory multiprocessor.

Section 8.1 describes the application of techniques for the symmetric tridiagonal eigenproblem including the bisection and inverse iteration routine B/III developed in Chapter 6 to the bidiagonal singular value problem. Sections 8.2 and 8.3 review methods that use the bidiagonal matrix itself. The first of these is a divide and conquer method based on rank one updating techniques and implemented as PSVD [46]. This method involves deleting a column from B and is a special case of the general rank one updates to the singular value decomposition described in [10]. An alternative method based on deleting a row of B and implicitly forming the matrix product $B^T B$ has been suggested by Arbenz and Golub [2]. The second method that works with bidiagonal matrices is the Golub-Reinsch implicit QR algorithm [35] used in the LINPACK code DSVDC [22].

Section 8.4. gives a serial comparison of B/III, PSVD, and DSVDC. Section 8.5 discusses the parallelism of the methods.

8.1 Solving the Bidiagonal Problem as a Tridiagonal One

While the bidiagonal singular value problem is closely related to the symmetric tridiagonal eigenvalue problem, care must be taken in applying the results of the preceding chapters to the computation of $B = Y\Sigma X^T$.

The matrix products $T_1 = B^T B$ and $T_2 = BB^T$ are symmetric tridiagonal matrices of order n having as eigenvalues the squares of the singular values of B and having as eigenvectors, respectively, the right and left singular vectors of B. Thus, one way to determine the SVD of B is to form T_1 and T_2 and compute the eigendecompositions $T_1 = X\Sigma^2 X^T$ and $T_2 = Y\Sigma^2 Y^T$.

Multiplication of B and its transpose in finite precision arithmetic, however, can lead to significant errors in small computed singular values [35]. For example, suppose that $\mathrm{fl}(1+\epsilon^2) = 1$ in finite precision arithmetic. If

$$B = \begin{pmatrix} 1 & 0 \\ 1 & \epsilon \end{pmatrix},$$

then the computed product $T_2 = BB^T$ is

$$\mathrm{fl}\left[\begin{pmatrix} 1 & 0 \\ 1 & \epsilon \end{pmatrix}\begin{pmatrix} 1 & 1 \\ 0 & \epsilon \end{pmatrix}\right] = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

with exact eigenvalues 0 and 2. The computed singular values of B are then 0 and $\sqrt{2}$, while the true singular values are $\left(\frac{2\epsilon^2}{2+\epsilon^2+\sqrt{4+\epsilon^4}}\right)^{\frac{1}{2}}$ and $\left(\frac{2+\epsilon^2+\sqrt{4+\epsilon^4}}{2}\right)^{\frac{1}{2}}$. For this matrix, the relative error in the smallest computed singular value is one.

One alternative is to embed the order n bidiagonal matrix in an order $2n$ symmetric tridiagonal matrix: the eigenvalues of the $2n \times 2n$ matrix

$$M_1 = \begin{pmatrix} 0 & B^T \\ B & 0 \end{pmatrix}$$

are the singular values of B and their negatives. If the columns and rows of M_1 are permuted to the order $1, n+1, 2, n+2, \ldots, n, 2n$, the resulting matrix M_2 is [32]

$$\begin{pmatrix} 0 & \alpha_1 & & & & \\ \alpha_1 & 0 & \beta_2 & & & \\ & \beta_2 & 0 & \alpha_2 & & \\ & & \ddots & \ddots & \ddots & \\ & & & & \alpha_n & 0 \end{pmatrix}.$$

For $1 \leq i \leq n$, the eigenvector u_i of M_2 corresponding to eigenvalue $\lambda_i = \sigma_i$ has as its odd-numbered components the components of the ith left singular vector $y_i = (\nu_{1i}, \ldots, \nu_{ni})^T$ and as its even-numbered components the components of the ith right singular vector $x_i = (\mu_{1i}, \ldots, \mu_{ni})^T$:

$$M_2 \begin{pmatrix} \nu_{1i} \\ \mu_{1i} \\ \vdots \\ \nu_{ni} \\ \mu_{ni} \end{pmatrix} = \sigma_i \begin{pmatrix} \nu_{1i} \\ \mu_{1i} \\ \vdots \\ \nu_{ni} \\ \mu_{ni} \end{pmatrix}.$$

Bisection with inverse iteration, the QL method, or Cuppen's method may then be applied directly to the matrix M_2.

When bisection and inverse iteration are used, the structure of the resulting tridiagonal matrix M_2 leads to a savings in computation time over that generally required for the order $2n$ symmetric tridiagonal eigenproblem. The Sturm sequence used for computing the eigenvalues of M_2 by bisection has $2n$ terms, but because all diagonal elements are zero, its evaluation requires only $2n - 1$ divisions and $2n - 1$ subtractions when the elements $\beta_2^2, \ldots, \beta_n^2$ are computed in advance. Thus, each Sturm sequence evaluation requires about $\frac{2}{3}$ times the number of floating point operations needed for an order $2n$ matrix with arbitrary diagonal elements. Moreover, only the n largest (nonnegative) eigenvalues of M_2 need be computed. The n corresponding eigenvectors determine Y and X. To obtain orthogonal singular vectors, the columns of Y and X should be reorthogonalized separately at each iteration, so the cost of computing the singular vector matrices is roughly the same as computing 2 $n \times n$ eigenvector matrices in the tridiagonal case.

The structure of M_2 does not lead to such significant savings for the other two methods. While the symmetric QL iteration [9] can be stopped after the n desired eigenpairs are computed, they may not be the first n produced [9]. In addition, the first factorization step in a shifted QL method fills in all diagonal elements, so beginning with a zero diagonal in M_2 leads to a savings of only $O(n)$

operations. Similarly, Cuppen's divide and conquer method [13] takes advantage of the zero diagonal only at the leaves of the computational tree. The diagonal elements used at subsequent levels depend on the eigenvalues and eigenvectors of the preceding subproblems and are not zero in general. Moreover, computing n of the $2n$ eigenpairs represents a savings only at the highest level.

These observations suggest that because bisection and inverse iteration are efficient for the symmetric tridiagonal eigenproblem, they should remain so for the bidiagonal singular value problem. Furthermore, the zero diagonal may give bisection and inverse iteration a further advantage over the QL and divide and conquer methods.

8.2 A Divide and Conquer Method for the Bidiagonal Singular Value Problem

This section presents a divide and conquer technique designed for use with the matrix B. It is an efficient alternative to Cuppen's method applied to a $2n \times 2n$ tridiagonal matrix. The divide and conquer method presented in this section avoids the numerical difficulties associated with explicit formation of BB^T or B^TB by implicitly converting the order n bidiagonal singular value problem to an order n symmetric tridiagonal eigenproblem. The algorithm relies on rank one tearing. Specifically, the rank one modification of B

$$B = \begin{pmatrix} B_1 & \beta e_k e_1^T \\ 0 & B_2 \end{pmatrix} = \begin{pmatrix} B_1 & 0 \\ 0 & B_2 \end{pmatrix} + \beta \begin{pmatrix} e_k \\ 0 \end{pmatrix} (0, e_1^T), \tag{8.1}$$

where $\beta = \beta_k$, allows implicit formation of BB^T as follows:

$$\begin{aligned} BB^T &= \begin{pmatrix} B_1 & \beta e_k e_1^T \\ 0 & B_2 \end{pmatrix} \begin{pmatrix} B_1^T & 0 \\ \beta e_1 e_k^T & B_2^T \end{pmatrix} \\ &= \begin{pmatrix} B_1 B_1^T & 0 \\ 0 & B_2(I - e_1 e_1^T) B_2^T \end{pmatrix} + \begin{pmatrix} \beta e_k \\ B_2 e_1 \end{pmatrix} (\beta e_k^T, e_1^T B_2^T). \end{aligned}$$

Moreover,

$$B_2(I - e_1 e_1^T) B_2^T \equiv \hat{B}_2 \hat{B}_2^T,$$

where

$$\hat{B}_2 = B_2 \begin{pmatrix} 0 & 0 \\ 0 & I \end{pmatrix}$$

is the bidiagonal matrix B_2 with its first column replaced by the zero vector, so that

$$BB^T = \begin{pmatrix} B_1 B_1^T & 0 \\ 0 & \hat{B}_2 \hat{B}_2^T \end{pmatrix} + \begin{pmatrix} \beta e_k \\ \alpha e_1 \end{pmatrix} (\beta e_k^T, \alpha e_1^T), \qquad (8.2)$$

with $\alpha e_1 = B_2 e_1$.

The singular value decompositions $B_1 = U_1 \Sigma_1 V_1^T$ and $\hat{B}_2 = \hat{U}_2 \hat{\Sigma}_2 \hat{V}_2^T$ can be computed independently and used with equation (8.2) to produce

$$\begin{aligned} BB^T &= \begin{pmatrix} U_1 \Sigma_1^2 U_1^T & 0 \\ 0 & \hat{U}_2 \hat{\Sigma}_2^2 \hat{U}_2^T \end{pmatrix} + \begin{pmatrix} \beta e_k \\ \alpha e_1 \end{pmatrix} (\beta e_k^T, \alpha e_1^T), \\ &= \begin{pmatrix} U_1 & 0 \\ 0 & \hat{U}_2 \end{pmatrix} \left[\begin{pmatrix} \Sigma_1^2 & 0 \\ 0 & \hat{\Sigma}_2^2 \end{pmatrix} + \begin{pmatrix} u_1 \\ \hat{u}_2 \end{pmatrix} (u_1^T, \hat{u}_2^T) \right] \begin{pmatrix} U_1^T & 0 \\ 0 & \hat{U}_2^T \end{pmatrix}, \qquad (8.3) \end{aligned}$$

where $u_1 = \beta U_1^T e_k$ and $\hat{u}_2 = \alpha \hat{U}_2^T e_1$. The eigendecomposition of the diagonal plus rank one matrix can be found via the techniques derived in [11, 24] and summarized in Chapter 4.

As in the tridiagonal case, this computation requires that the diagonal elements of the matrix

$$\begin{pmatrix} \Sigma_1^2 & 0 \\ 0 & \hat{\Sigma}_2^2 \end{pmatrix}$$

be distinct and that the elements of $(u_1^T, \hat{u}_2^T)$ be nonzero. When these assumptions do not hold, the problem deflates. However, because the squares of the singular values less than one are not as well-separated as the singular values themselves, the deflation rules of [24] described in Chapter 4 concerning nearly equal singular values are not appropriate. To develop deflation rules for the SVD, it is necessary to reformulate the basic step of the updating process and, thereby, provide rules based on the original data rather than on the squared data appearing in equation (8.3).

To this end, let the $(n-k) \times (n-k-1)$ matrix $\tilde{B}_2$ be defined by

$$(0, \tilde{B}_2) \equiv \hat{B}_2 \equiv B_2 \left(I - e_1 e_1^T \right).$$

Now consider the singular value decomposition of $\tilde{B}_2 = \tilde{U}_2 \tilde{\Sigma}_2 \tilde{V}_2^T$, and note that

$$\hat{B}_2 = \begin{pmatrix} \tilde{u} & \tilde{U}_2 \end{pmatrix} \begin{pmatrix} 0 & 0 \\ 0 & \tilde{\Sigma}_2 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & \tilde{V}_2^T \end{pmatrix},$$

where $\tilde{u}$ is a unit vector orthogonal to the columns of $\tilde{U}_2$. Using the tearing of equation (8.1),

$$B = \begin{pmatrix} U_1 & 0 & 0 \\ 0 & \tilde{u} & \tilde{U}_2 \end{pmatrix} \begin{pmatrix} \Sigma_1 & u_1 & 0 \\ 0 & \mu & 0 \\ 0 & u_2 & \tilde{\Sigma}_2 \end{pmatrix} \begin{pmatrix} V_1^T & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \tilde{V}_2^T \end{pmatrix}, \tag{8.4}$$

where

$$\begin{pmatrix} U_1 & 0 & 0 \\ 0 & \tilde{u} & \tilde{U}_2 \end{pmatrix} \begin{pmatrix} u_1 \\ \mu \\ u_2 \end{pmatrix} = \begin{pmatrix} \beta e_k \\ \alpha e_1 \end{pmatrix}.$$

For notational convenience [1], we permute equation (8.4) to obtain

$$B = \begin{pmatrix} U_1 & 0 & 0 \\ 0 & \tilde{U}_2 & \tilde{u} \end{pmatrix} \begin{pmatrix} \Sigma_1 & 0 & u_1 \\ 0 & \tilde{\Sigma}_2 & u_2 \\ 0 & 0 & \mu \end{pmatrix} \begin{pmatrix} V_1^T & 0 & 0 \\ 0 & 0 & \tilde{V}_2^T \\ 0 & 1 & 0 \end{pmatrix}. \tag{8.5}$$

Deflation rules are then needed for the interior matrix

$$\bar{M} \equiv \begin{pmatrix} \bar{\Sigma} & \bar{u} \\ 0 & \mu \end{pmatrix} \equiv \begin{pmatrix} \Sigma_1 & 0 & u_1 \\ 0 & \tilde{\Sigma}_2 & u_2 \\ 0 & 0 & \mu \end{pmatrix}$$

where $\bar{\Sigma} = \mathrm{diag}(\bar{\sigma}_1, \ldots, \bar{\sigma}_{n-1})$ and $\bar{u} = (\bar{\mu}_1, \ldots, \bar{\mu}_{n-1})$.

The deflation procedure for $\bar{M}$ resembles that for tridiagonal matrices. The rules for exact arithmetic follow. If $\bar{\mu}_j$ is zero, $\bar{\sigma}_j$ is a singular value of $\bar{M}$ with left and right singular vectors equal to e_j. If $\bar{\sigma}_i = \bar{\sigma}_j$ for some $i \neq j$ or if $\bar{\sigma}_j = 0$, plane rotations are applied to reduce $\bar{\mu}_j$ to zero so that $\bar{M} e_j = \bar{\sigma}_j e_j$ [46]. In particular, when $\bar{\sigma}_i = \bar{\sigma}_j$, two-sided rotations are applied as in the tridiagonal case:

$$\begin{pmatrix} \gamma & \sigma \\ -\sigma & \gamma \end{pmatrix} \left[\begin{pmatrix} \bar{\sigma}_i & \\ & \bar{\sigma}_j \end{pmatrix} + \begin{pmatrix} \bar{\mu}_i \\ \bar{\mu}_j \end{pmatrix} \begin{pmatrix} \bar{\mu}_i, & \bar{\mu}_j \end{pmatrix} \right] \begin{pmatrix} \gamma & -\sigma \\ \sigma & \gamma \end{pmatrix}$$
$$= \begin{pmatrix} \bar{\sigma}_i' & \\ & \bar{\sigma}_j' \end{pmatrix} + \begin{pmatrix} \bar{\mu}_i' \\ 0 \end{pmatrix} \begin{pmatrix} \bar{\mu}_i', & 0 \end{pmatrix},$$

[1] The matrices are not explicitly permuted in the implementation of PSVD.

with $\gamma^2 + \sigma^2 = 1$ and $(\mu_i')^2 = \bar{\mu}_i^2 + \bar{\mu}_j^2$. When $\bar{\sigma}_j = 0$, a one-sided rotation suffices when it uses $\mu = e_n^T \bar{M} e_n$ to zero out $\bar{\mu}_j$:

$$\begin{pmatrix} \gamma & \sigma \\ -\sigma & \gamma \end{pmatrix} \left[\begin{pmatrix} 0 & \\ & 0 \end{pmatrix} + \begin{pmatrix} \bar{\mu}_j \\ \mu \end{pmatrix} (\, \bar{\mu}_j, \quad \mu \,) \right] = \begin{pmatrix} 0 & \\ & 0 \end{pmatrix} + \begin{pmatrix} 0 \\ \bar{\mu}' \end{pmatrix} (\, 0, \quad \bar{\mu}_j' \,),$$

with $\gamma > 0$ [46].

After deflation, one need only compute the singular value decomposition of

$$M \equiv \begin{pmatrix} \tilde{\Sigma} & u \\ 0 & \mu \end{pmatrix} \equiv Y \Sigma X^T,$$

where $\tilde{\Sigma}$ has distinct, positive elements and the vector u has only nonzero elements. The squares of the singular values and the left singular vectors are given by the eigendecomposition

$$Y \Sigma^2 Y^T \equiv M M^T \equiv \begin{pmatrix} \tilde{\Sigma}^2 & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} u \\ \mu \end{pmatrix} (u^T, \mu). \qquad (8.6)$$

An eigenvalue σ^2 of MM^T is a root of the secular equation, *i.e.*,

$$1 + u^T (\tilde{\Sigma}^2 - \sigma^2)^{-1} u - \left(\frac{\mu}{\sigma} \right)^2 = 0$$

and can be computed using the root-finder from [11] described in Chapter 4. No further deflation is needed. If the sorted diagonal elements of $\mathrm{diag}(\tilde{\Sigma}^2, 0)$ are $0 = \tilde{\sigma}_1^2 < \tilde{\sigma}_2^2 < \ldots < \tilde{\sigma}_n^2$, the jth eigenvalue σ_j^2 of MM^T lies in the interval $(\tilde{\sigma}_j^2, \tilde{\sigma}_{j+1}^2)$ [11] so that all eigenvalues are positive. The jth singular value of B is σ_j, and the left singular vector of M associated with σ_j for $j = 1, \ldots, n$ is

$$y_j = \begin{pmatrix} (\tilde{\Sigma}^2 - \sigma^2)^{-1} u \\ -\mu / \sigma_j^2 \end{pmatrix} \theta,$$

where θ is a normalization factor. The corresponding right singular vector is

$$x_j = \frac{M^T y_j}{\| M^T y_j \|_2}.$$

The singular values of B are those of $\bar{M}$. The left and right singular vectors of B are those of $\bar{M}$ premultiplied by, respectively,

$$\begin{pmatrix} U_1 & 0 & 0 \\ 0 & \tilde{U}_2 & \tilde{u} \end{pmatrix}$$

and

$$\begin{pmatrix} V_1^T & 0 & 0 \\ 0 & 0 & \tilde{V}_2^T \\ 0 & 1 & 0 \end{pmatrix}^T .$$

This algorithm has been implemented as PSVD using finite-precision versions of these deflation rules [46].

8.3 The Golub-Reinsch QR Algorithm

The Golub-Reinsch QR algorithm is an iterative method for reducing B to diagonal form through orthogonal transformations. The basic step is an algorithm developed in [32]. If $B = Y\Sigma X^T$, the orthogonal matrices Y and X are formed simultaneously by implicitly forming the symmetric tridiagonal matrix B^TB and applying the symmetric QR algorithm [35, 33]. (The QL algorithm for the symmetric tridiagonal eigenproblem is described in Chapter 4.)

The kth iterate is a bidiagonal matrix $B_k = Y_kB_{k-1}X_k^T$, where, as in the symmetric case, the orthogonal matrices Y_k and X_k are products of plane rotations. As k approaches infinity, the iterates B_k converge to a diagonal matrix with the singular values of B on its diagonal. The accumulated product $Y_1^T \dots Y_k^T$ is the transposed matrix of left singular vectors of B and $X_1^T \dots X_k^T$ the right singular vectors of B.

The Golub-Reinsch QR algorithm has been implemented as the LINPACK routine DSVDC [22]. DSVDC is based on the implicit-shift QR method but uses a variant of the Wilkinson shift which is easier to compute than the conventional eigenvalue of the trailing 2×2 submatrix of B^TB. The shift is the eigenvalue of the trailing 2×2 submatrix of BB^T closer to the last diagonal element of BB^T [22]. An off-diagonal element $\hat{\beta}_m^{(k)}$ of iterate B_k is considered negligible if

$$|\hat{\beta}_m^{(k)}| \leq \epsilon_M(|\hat{\alpha}_{m+1}^{(k)}| + |\hat{\alpha}_m^{(k)}|),$$

where $\hat{\alpha}_m^{(k)}$ and $\hat{\alpha}_{m+1}^{(k)}$ are the diagonal elements of B_k adjacent to $\hat{\beta}_m^{(k)}$. If the last off-diagonal element of B_k is negligible, its last diagonal element is a computed singular value of B. If any other off-diagonal element of B_k is negligible, the matrix splits at that point, and iteration continues with the leading unreduced submatrix [22].

8.4 Serial Experiments

The same performance indicators defined for the tridiagonal case apply to the bidiagonal problem. Values of each for the five test problems are given in Table 8.1.

matrix	order n	PSVD: roots computed (scaled)	DSVDC: $\mathcal{N} = \frac{1}{n}\sum_{i=1}^{m} n_i$	B/III: fraction of BISECT time
$[2,1]$	32	1.0	40	.75
	100	2.7	113	.72
	200	3.3	210	.71
random	32	1.0	34	.73
	100	2.2	111	.72
	200	3.0	213	.71
$[2,u]/n$	32	0.9	36	.76
	100	2.0	103	.74
	200	2.1	177	.73
B_W	32	0.6	33	.76
	100	1.7	97	.62
	200	1.9	180	.58
modified $[2,1]$	32	1.0	41	.71
	100	2.7	107	.71
	200	3.3	205	.70

Table 8.1: The number of roots computed by PSVD divided by the matrix order, the order index for DSVDC, and the fraction of time spent in BISECT by B/III (with implicit conversion to tridiagonal form) for five bidiagonal matrices.

The total number of roots computed by PSVD at all but the lowest level divided by the matrix order measures the deflation in the problem. PSVD forms computational trees of height 2 for $n = 32$, 4 for $n = 100$, and 5 for $n = 200$. At each level, a total of n eigenpairs is computed. Thus, respective scaled root counts of 1.0, 3.0, and 4.0 for the three orders indicate that no deflation has occurred. Matrices $[2, u]/n$ and B_W exhibit the most deflation and matrices [2,1] and modified [2,1] the least.

Matrix splitting in DSVDC is measured by $\mathcal{N} = \sum_{i=1}^{m} n_i$, where n_i is the submatrix at iteration i and m is the total number of iterations. Matrices $[2, u]/n$ and B_W split somewhat more than do the others.

The ratios of BISECT and III times are given in the last column. BISECT occupies roughly $\frac{7}{10}$ of the total time for all matrices except B_W whose clustered singular values lower the ratio. The correlation between clustering and BISECT time is given in Table 8.2. None of the test matrices has clusters of more than four singular values, hence none spends appreciable time in reorthogonalization by the Modified Gram-Schmidt procedure. Clustered singular values do, however, lower both the fraction of time spent in BISECT and the total time spent in B/III.

The performance measures are reflected in the data given in Figures 8.1–8.5 and Tables 8.3 and 8.4. For orders over about 50, the results resemble those for the symmetric tridiagonal case. The QR method DSVDC is always the slowest and the small variations in $\mathcal{N}$ for the test problems have little influence on the runtimes. The relative times for B/III and PSVD depend largely on the amount of deflation in PSVD. PSVD is fastest only for large order problems with significant deflation (B_W and $[2, u]/n$). Singular value clustering lowers the time for B/III for B_W relative to the other matrices.

The low order results depicted in the top graphs of Figures 8.1–8.5 differ from those for the tridiagonal case. DSVDC is often fastest for problems of order less than or equal to 20. Divide and conquer is consistently the fastest technique for all problems of orders between 20 and 60. B/III is the slowest method for all problem orders up to 50. (This slowness is not caused by any obvious coding inefficiency, although the runtime is dominated by BISECT in all cases.)

matrix	order n	number of clustered eigenvalues	maximum cluster size	time for B/III (seconds)	time ratios	
					$\frac{\textsc{Bisect}}{\textsc{Total}}$	$\frac{\textsc{Mgs}}{\textsc{Total}}$
$[2,1]$	32	0	0	.73	.75	0
	100	0	0	6.05	.72	0
	200	0	0	22.85	.71	0
random	32	0	0	.73	.73	0
	100	0	0	6.06	.72	.01
	200	0	0	22.82	.71	.01
$[2,u]/n$	32	0	0	.75	.76	0
	100	0	0	6.07	.74	0
	200	1	2	22.81	.73	0
B_W	32	9	2	.62	.76	0
	100	43	2	4.45	.62	0
	200	94	2	16.35	.58	.03
modified $[2,1]$	32	4	4	.71	.71	0
	100	4	4	6.00	.71	0
	200	4	4	22.60	.70	0

Table 8.2: Number of singular values with spacing less than $10^{-14}\| B \|_2$, maximum cluster size, and fractions of B/III times spent in BISECT and MGS.

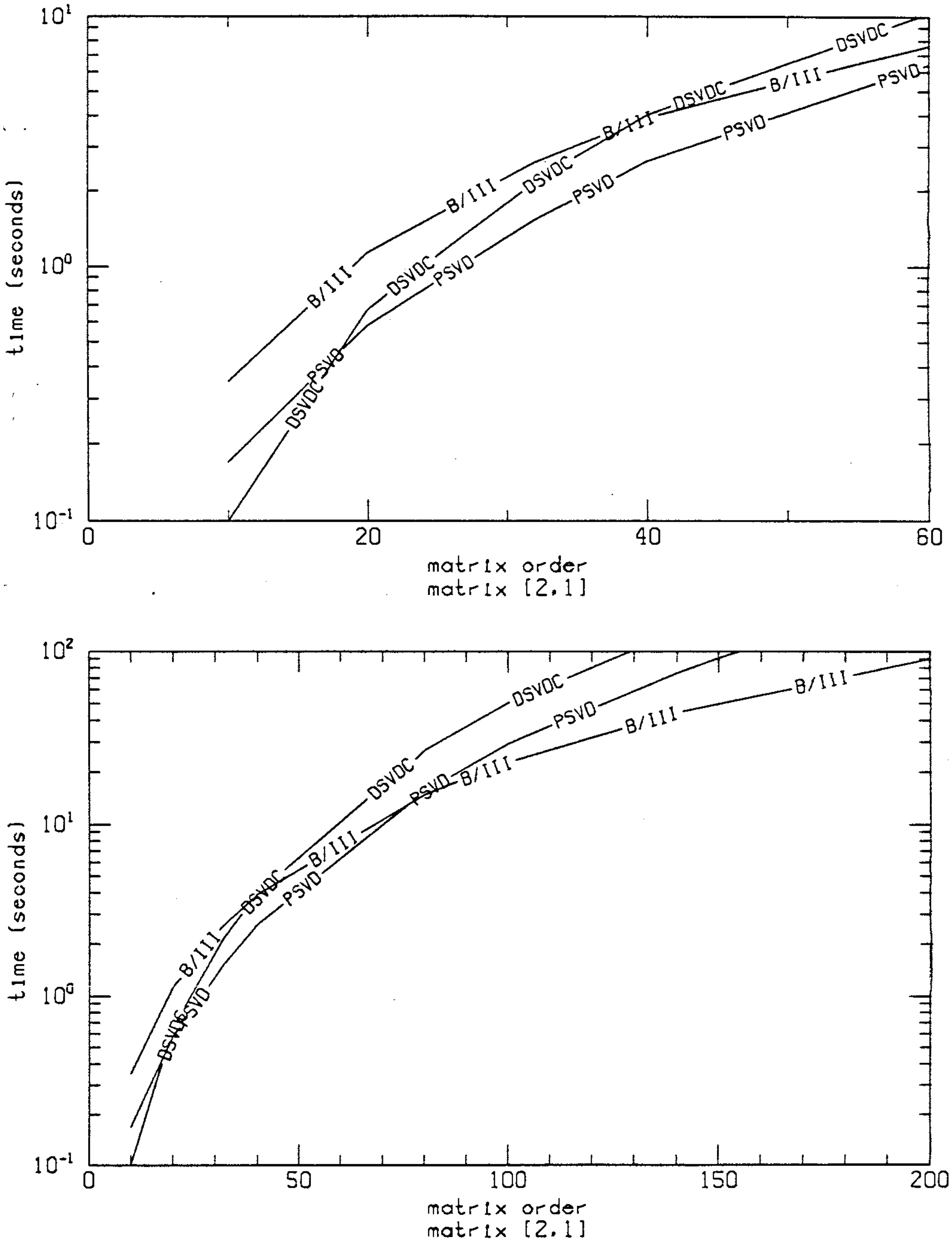

Figure 8.1: Times for computation of the SVD by B/III, PSVD, and DSVDC *versus* matrix order for matrix [2,1].

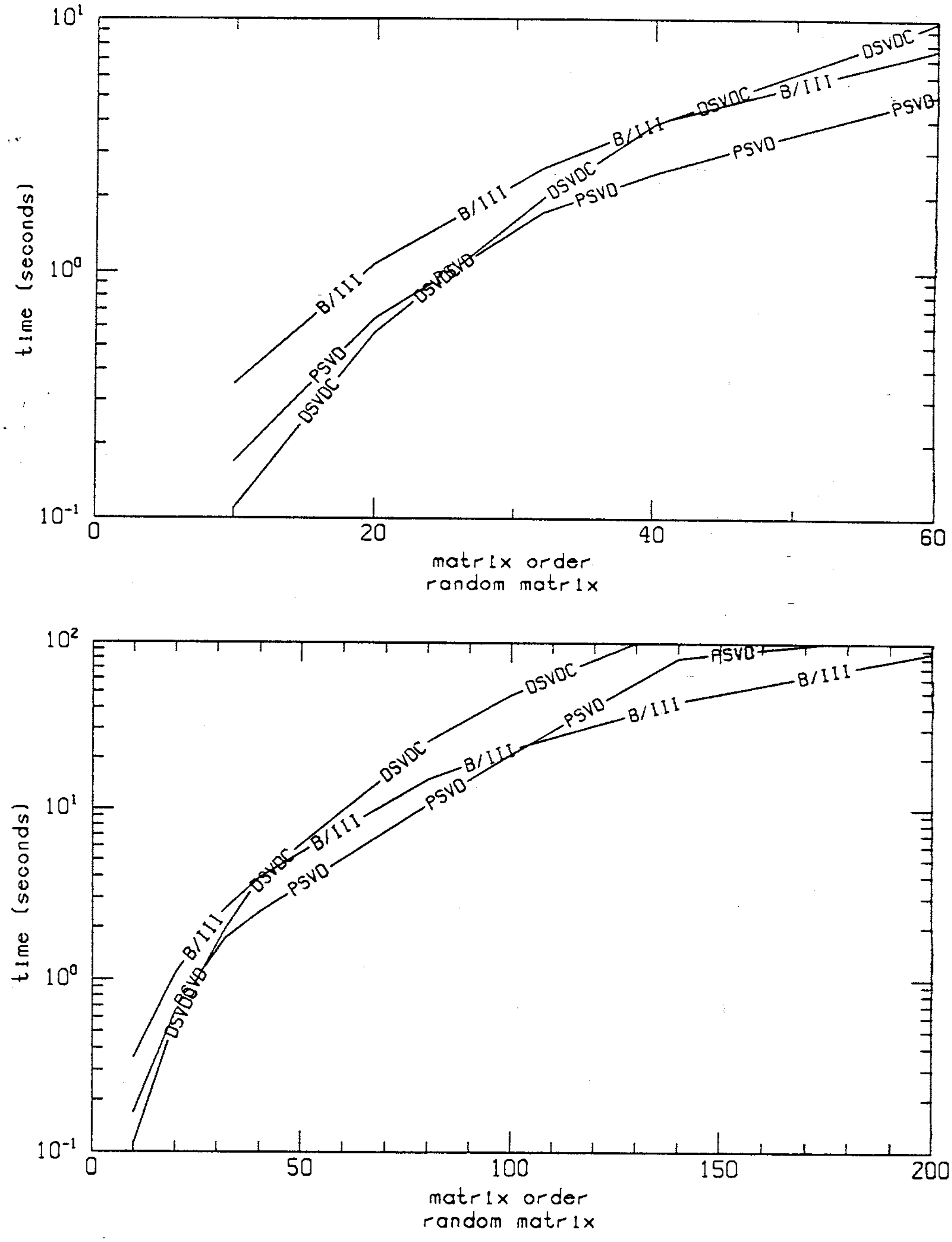

Figure 8.2: Times for computation of the SVD by B/III, PSVD, and DSVDC *versus* matrix order for random matrices.

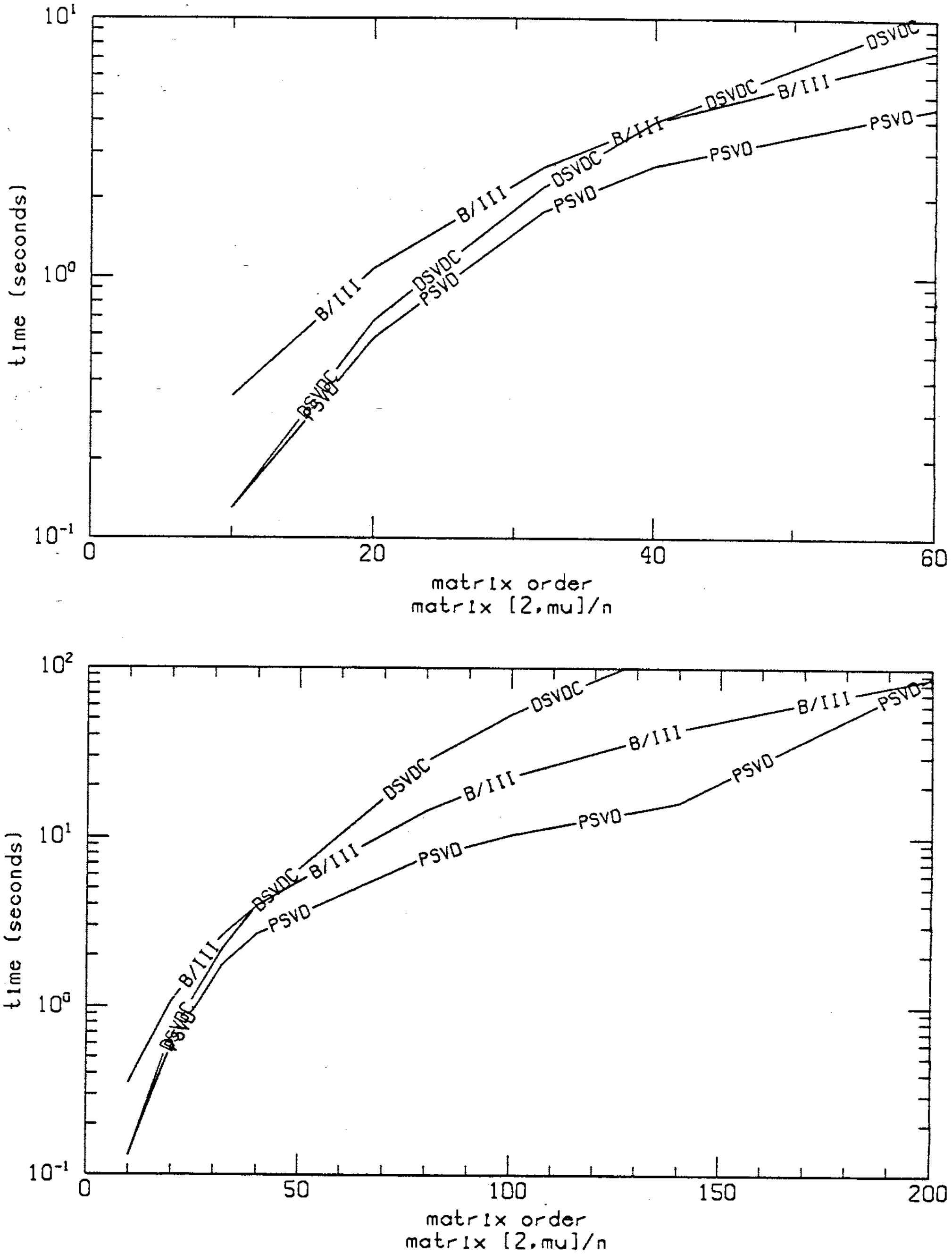

Figure 8.3: Times for computation of the SVD by B/III, PSVD, and DSVDC *versus* matrix order for matrix [2,u]/n.

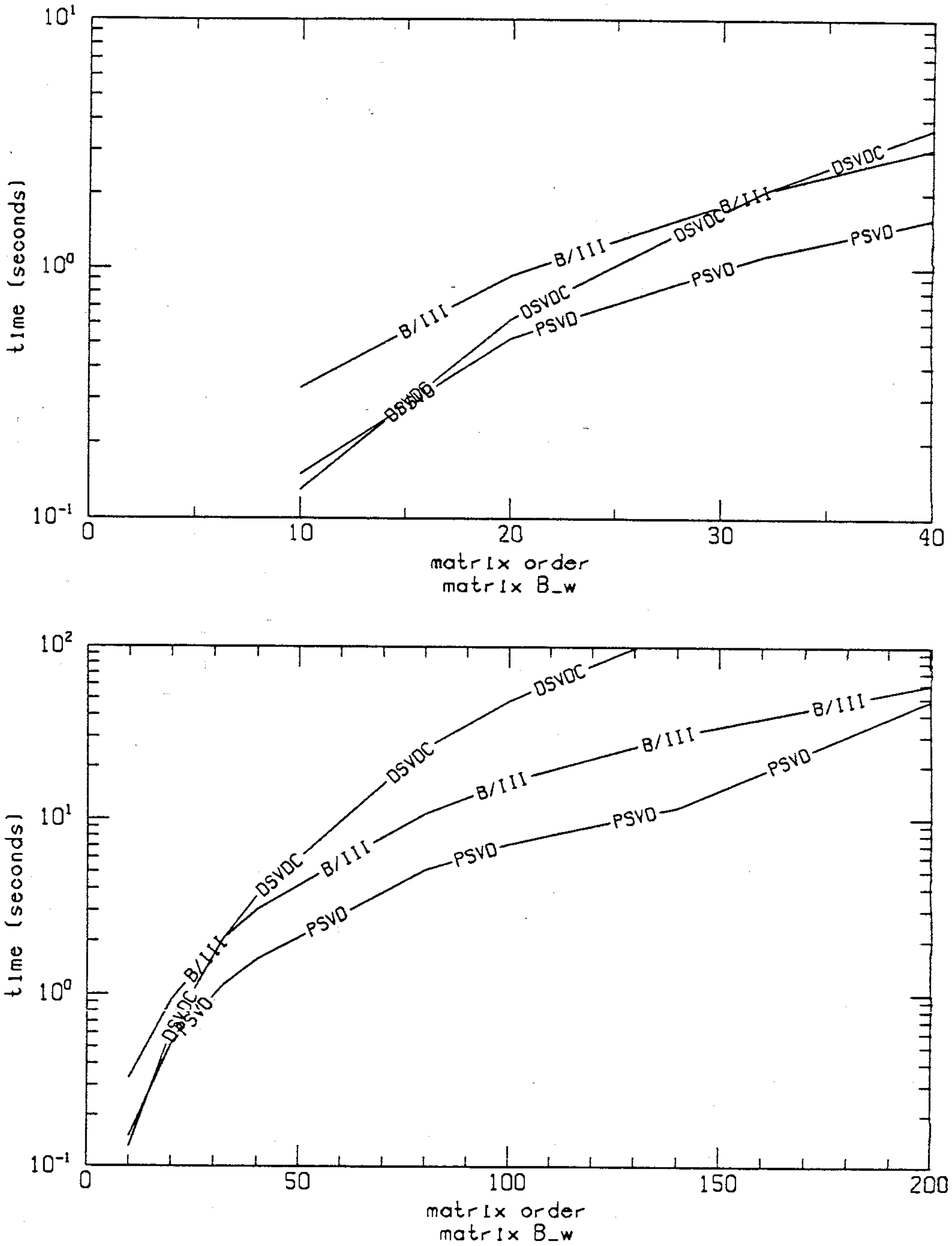

Figure 8.4: Times for computation of the SVD by B/III, PSVD, and DSVDC *versus* matrix order for matrix B_W.

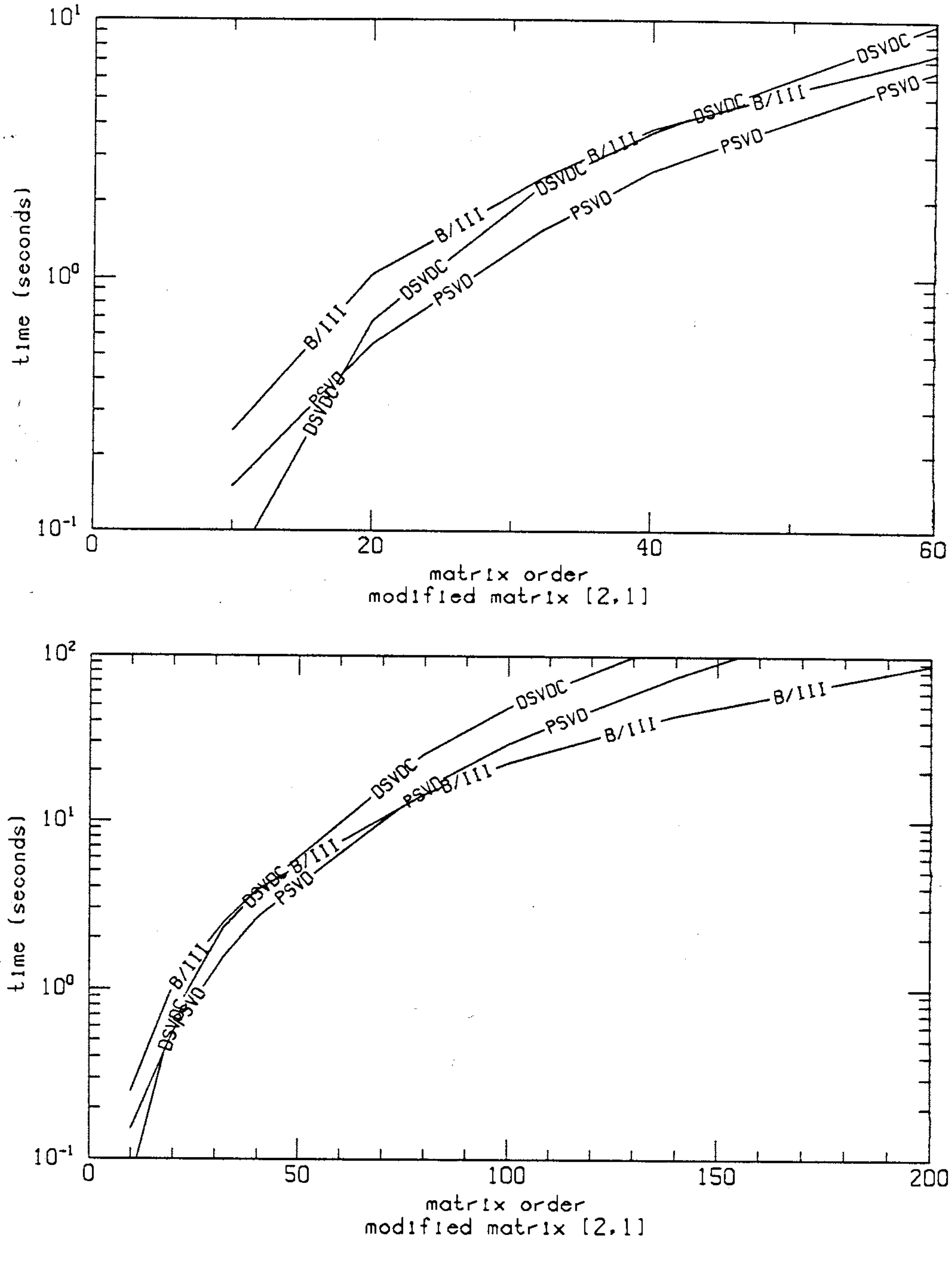

Figure 8.5: Times for computation of the SVD by B/III, PSVD, and DSVDC *versus* matrix order for the modified matrix [2,1].

matrix	order	time for	time for	time for
		B/III	PSVD	DSVDC
		(seconds)	(seconds)	(seconds)
$[2,1]$	10	.35	.17	.10
	20	1.13	.58	.67
	32	2.60	1.53	2.17
	40	3.93	2.63	4.03
	80	14.83	15.52	26.63
	100	22.85	29.37	50.13
	140	44.13	74.27	130.13
	200	89.95	255.22	359.15
random	10	.35	.17	.11
	20	1.08	.65	.57
	32	2.60	1.73	1.97
	40	3.93	2.47	3.87
	80	14.87	10.53	25.00
	100	22.55	21.13	47.83
	140	43.77	80.01	129.27
	200	87.75	117.83	361.10
$[2,u]/n$	10	.35	.13	.13
	20	1.07	.58	.68
	32	2.63	1.77	2.20
	40	3.95	2.67	3.95
	80	14.43	7.65	27.93
	100	22.88	10.30	53.33
	140	43.50	16.03	135.80
	200	87.27	88.60	366.93

Table 8.3: Times for computation of the SVD by B/III, PSVD, and DSVDC for matrix [2,1], random matrices, and matrix $[2,u]/n$.

matrix	order	time for B/III (seconds)	time for PSVD (seconds)	time for DSVDC (seconds)
B_W	10	.33	.15	.13
	20	.93	.52	.62
	32	2.05	1.12	2.05
	40	3.05	1.58	3.67
	80	10.65	5.10	25.55
	100	16.35	7.15	47.87
	140	30.52	11.60	125.72
	200	60.25	48.60	342.28
modified [2, 1]	10	.25	.15	.07
	20	1.03	.55	.68
	32	2.45	1.52	2.28
	40	3.83	2.63	3.70
	80	14.48	15.52	25.40
	100	22.45	29.42	48.65
	140	43.50	74.18	126.68
	200	88.82	255.62	354.15

Table 8.4: Times for computation of the SVD by B/III, PSVD, and DSVDC for matrix B_W and the modified matrix [2,1].

matrix order	method	maximum residual $\mathcal{R}$	maximum orthogonality (left) $\mathcal{O}_Y$	maximum orthogonality (right) $\mathcal{O}_X$
n = 32				
	PSVD	1.66d-14	7.65d-15	7.54d-15
	DSVDC	1.79d-15	1.13d-14	1.13d-14
	B/III	9.77d-16	9.76d-15	1.02d-14
n = 100				
	PSVD	9.39d-14	2.56d-14	2.37d-14
	DSVDC	4.22d-15	2.68d-14	2.86d-14
	B/III	2.38d-15	1.90d-14	1.87d-14
n = 200				
	PSVD	4.09d-15	1.13d-14	1.64d-14
	DSVDC	7.60d-15	8.13d-14	8.14d-14
	B/III	5.99d-15	5.42d-14	5.53d-14

Table 8.5: Maximum residual and orthogonalities of singular value decompositions computed by B/III, PSVD, and DSVDC for the five test matrices.

Table 8.5 lists the maximum residual $\mathcal{R}$ and orthogonality $\mathcal{O}$ recorded for any of the test problems when $n = 32$, 100, or 200. All methods produce results to the same high accuracy when small singular values are computed.

8.5 Parallelism

The implementation of B/III for the bidiagonal SVD on a statically-scheduled, distributed-memory multiprocessor is essentially the same as for the symmetric tridiagonal eigenproblem (*cf.* Chapter 5). The only difference is that the left and right singular vectors must be reorthogonalized independently. Similarly, parallel implementation of DSVDC proceeds as for TQL2 with rotations accumulated separately to form the left and right singular vectors. Thus, both B/III for the SVD and DSVDC have a maximum theoretical speedup of about p on a p-processor hypercube. For B/III, this maximum can be approached only when processor workloads are well-balanced.

The parallelism of PSVD can be estimated through a step-by-step comparison with the divide and conquer code TREEQL for the symmetric tridiagonal eigenproblem. Table 8.6 outlines the work done in a subcube of dimension j when solving the order $k2^j$ subproblem at updating step j by PSVD and by TREEQL. First, the subproblem is split into two order $k2^{j-1}$ subproblems. These problems are solved in parallel, one problem per subcube of dimension $j-1$.

Second, the elements of D and z or of $\bar{\Sigma}$ and $(\bar{u}^T, \bar{\mu})$ are distributed among the processors of the subcube by an alternate direction exchange by Algorithm 3.3.1. Processors participating in the exchange are thus synchronized at this point.

Third, deflation (or the absence thereof) is identified by each processor at entry 3. In both algorithms, this represents a serial bottleneck as all processors determine and apply the same rotations for deflation.

Fourth, each processor computes its share of the eigenpairs of the deflated problems. On a statically-scheduled hypercube, processor loads can be severely imbalanced at this point. The advantage of deflation is thus lost for both the symmetric tridiagonal eigenproblem and the bidiagonal SVD.

Fifth, the computed eigenvectors are updated. On the hypercube, this can be done using Algorithm 3.3.2. For the SVD, the updating matrices, which are in block form, can be compacted into a single dense matrix thereby saving some startup costs. The processors are synchronized by the distributed matrix multiplication.

The times for Algorithms 3.3.1 and 3.3.2 are determined by the matrix order. The times to form and solve the deflated problem, on the other hand, depend on the degree of deflation. The experiments of Chapter 5 show that a maximum speedup of about $\frac{9}{10}p$ can be achieved for TREEQL for large problems with

Updating Step j in the Divide and Conquer Methods	
TREEQL: $T = \begin{pmatrix} T_1 & \alpha e_k e_1^T \\ \alpha e_1 e_k^T & T_2 \end{pmatrix} = U \Lambda U^T,$ T is of order $k2^j$.	PSVD: $B = \begin{pmatrix} B_1 & \beta e_k e_1^T \\ & B_2 \end{pmatrix} = U \Sigma V^T,$ B is of order $k2^j$.
1. Solve $T_1 = U_1 \Lambda_1 U_1^T$, $T_2 = U_2 \Lambda_2 U_2^T$. 2. Form $D + \rho z z^T$. 3. Deflate to form $\tilde{D} + \rho \tilde{z} \tilde{z}^T$. 4. Compute $\tilde{D} + \rho \tilde{z} \tilde{z}^T = Y \Lambda Y^T$. 5. Multiply $\begin{pmatrix} U_1 & \\ & U_2 \end{pmatrix} Y$.	1. Solve $B_1 = Y_1 \Sigma_1 X_1^T$, $\hat{B}_2 = Y_2 \Sigma_2 X_2^T$. 2. Form $\begin{pmatrix} \bar{\Sigma} & \bar{u} \\ 0 & \mu \end{pmatrix} = \bar{M}$. 3. Deflate to form $\tilde{M}$. 4. Compute $\tilde{M} = Y \Sigma^2 X^T$. 5. Multiply $\begin{pmatrix} Y_1 & & \\ & y & Y_2 \end{pmatrix} Y$ and $\begin{pmatrix} X_1 & & \\ & 1 & \\ & & X_2 \end{pmatrix} X$.

Table 8.6: Solution of a problem of order $k2^j$ on a j-cube.

almost no deflation but that the speedup falls to about $\frac{1}{2}p$ for problems with a moderate degree of deflation. Even lower speedups can be expected for problems where significant deflation occurs. A similar result can be anticipated for PSVD.

The experiments of Section 8.4, however, show that deflation is less prevalent for PSVD than for TREEQL. Thus, a generally higher speedup could be expected for PSVD. However, PSVD is only competitive with B/III on a single processor when deflation is significant. This combination of results suggests that bisection with inverse iteration remains the preferred algorithm on the statically-scheduled, distributed-memory multiprocessor.

Bibliography

[1] M. Abramowitz and I. Stegun, *Handbook of Mathematical Functions*, Dover Publications, Inc., 1965.

[2] P. Arbenz and G. Golub, *On the spectral decomposition of Hermitian matrices subjected to indefinite low rank perturbations*, Manuscript NA-87-07, Computer Science Dept, Stanford University, 1987.

[3] J. Barlow and J. Demmel, *Computing accurate eigensystems of scaled diagonally dominant matrices*, report 421, Computer Science Dept, Courant Institute, 1988.

[4] R. Barlow and D. Evans, *A parallel organization of the bisection algorithm*, The Computer Journal, 22 (1977), pp. 267–69.

[5] R. Barlow, D. Evans, and J. Shanehchi, *Parallel multisection applied to the eigenvalue problem*, The Computer Journal, 26 (1983), pp. 6–9.

[6] W. Barth, R. Martin, and J. Wilkinson, *Calculation of the eigenvalues of a symmetric tridiagonal matrix by the method of bisection*, in Handbook for Automatic Computation: Linear Algebra, Springer Verlag, 1971, pp. 249–256.

[7] K.-J. Bathe and E.L.Wilson, *Numerical Methods in Finite Element Analysis*, Prentice Hall, 1976.

[8] H. Bernstein and M. Goldstein, *Parallel implementation of bisection for the calculation of eigenvalues of tridiagonal symmetric matrices*, Computing, 37 (1986), pp. 85–91.

[9] H. Bowdler, R. Martin, and J. Wilkinson, *The QR and QL algorithms for symmetric matrices*, Numer. Math., 11 (1968), pp. 227–240.

[10] J. Bunch and C. Nielsen, *Updating the singular value decomposition*, Numer. Math., 31 (1978), pp. 111–129.

[11] J. Bunch, C. Nielsen, and D. Sorensen, *Rank-one modification of the symmetric eigenproblem*, Numer. Math., 31 (1978), pp. 31–48.

[12] S. Crump, *The estimation of variance components in analysis of variance*, Biometrics, 2 (1946), pp. 7–11.

[13] J. Cuppen, *A divide and conquer method for the symmetric tridiagonal eigenproblem*, Numer. Math., 36 (1981), pp. 177–95.

[14] E. de Doncker, J. Kapenga, and P. Dewilde, *The symmetric tridiagonal eigenproblem on a custom linear array and hypercubes*, Technical Report, Western Michigan University, 1988.

[15] P. Deift, J. Demmel, L.-C. Li, and C. Tomei, *LAPACK working note #11: The bidiagonal singular value decomposition and Hamiltonian mechanics*, Computer Science Dept. Technical Report, Courant Institute, 1989.

[16] P. Deift, T. Nanda, and C. Tomei, *Ordinary differential equations and the symmetric eigenvalue problem*, SIAM J. Numer. Anal., 20 (1983), pp. 1–22.

[17] J. Demmel and A. Greenbaum. Personal Communication, 1989.

[18] J. Demmel and W. Kahan, *LAPACK working note #3: Computing small singular values of bidiagonal matrices with guaranteed relative accuracy*, Mathematics and Computer Science Division, Argonne National Laboratory, 1988.

[19] J. Demmel and K. Veselić, *LAPACK working note #15: Jacobi's method is more accurate than qr*, Computer Science Dept. Technical Report, Courant Institute, 1989.

[20] L. Devroye, *Non-Uniform Random Variate Generation*, Springer-Verlag, 1986.

[21] J. Dixon, *Estimating extremal eigenvalues and condition numbers of matrices*, SIAM J. Numer. Anal., 20 (1983), pp. 812–814.

[22] J. Dongarra, J. Bunch, C. Moler, and G. Stewart, *LINPACK Users' Guide*, SIAM Publications, 1979.

[23] J. Dongarra, S. Hammarling, and D. Sorensen, *LAPACK working note #2: Block reduction to tridiagonal and Hessenberg form for the eigenvalue problem*, Mathematics and Computer Science Division, Argonne National Laboratory, 1987.

[24] J. Dongarra and D. Sorensen, *A fully parallel algorithm for the symmetric eigenvalue problem*, SIAM J. Sci. Stat. Comput., 8 (1987), pp. s139–s154.

[25] A. Dubrulle, R. Martin, and J. Wilkinson, *The implicit QL algorithm*, in Handbook for Automatic Computation: Linear Algebra, Springer Verlag, 1971, pp. 241–248.

[26] S. Eisenstat. Personal Communication, June, 1989.

[27] G. Forsythe, M. Malcolm, and C. Moler, *Computer Methods for Mathematical Computations*, Prentice Hall, 1977.

[28] G. Fox, A. Hey, and S. Otto, *Matrix algorithms on the hypercube I: Matrix multiplication*, Technical Report, California Institute of Technology, 1985.

[29] E. GILBERT, *Gray codes and paths on the n-cube*, The Bell System Technical Journal, (May 1958).

[30] W. GIVENS, *Numerical computation of the characteristic values of a real symmetric matrix*, Tech. Report ORNL-1574, Oak Ridge National Laboratory, 1954.

[31] G. GOLUB, *Some modified matrix eigenvalue problems*, SIAM Review, 15 (1973), pp. 318–34.

[32] G. GOLUB AND W. KAHAN, *Calculating the singular values and pseudo-inverse of a matrix*, J. SIAM Numer. Anal., Ser. B, Vol. 2 (1965), pp. 205–224.

[33] G. GOLUB AND C. V. LOAN, *Matrix Computations*, The Johns Hopkins Press, Baltimore, MD, 1983.

[34] G. GOLUB AND C. REINSCH, *Singular value decomposition and least squares solutions*, Numer. Math., 14 (1970), pp. 403–20.

[35] G. GOLUB AND C. REINSCH, *Singular value decomposition and least squares solutions*, in Handbook for Automatic Computation: Linear Algebra, Springer Verlag, 1971, pp. 134–151.

[36] R. GREGORY AND D. KARNEY, *A Collection of Matrices for Testing Computational Algorithms*, John Wiley and Sons, Inc., 1969.

[37] D. HELLER AND I. IPSEN, *Systolic networks for orthogonal decompositions*, SIAM J. Sci. Stat. Comp., 4 (1983), pp. 261–9.

[38] H. HUANG, *A parallel algorithm for symmetric tridiagonal eigenvalue problems*, CAC Document No. 109, Center for Advanced Computation, University of Illinois, 1974.

[39] I. IPSEN, *Singular value decomposition with systolic arrays*, in Proc. SPIE Symp. 549 (Real Time Signal Processing VII), 1984, pp. 13–21.

[40] I. Ipsen and E. Jessup, *Solving the symmetric tridiagonal eigenvalue problem on the hypercube*, Research Report 548, Dept. Computer Science, Yale University, 1987. To appear in SIAM J. Sci. Stat. Comput.

[41] ——, *Two methods for solving the symmetric tridiagonal eigenvalue problem on the hypercube*, in Hypercube Multiprocessors 1987, SIAM, 1987, pp. 627–638.

[42] E. Isaacson and H. Keller, *Analysis of Numerical Methods*, John Wiley and Sons, 1966.

[43] R. Jensen. Personal Communication, January, 1985.

[44] ——, *Chaos in atomic physics*, in Proceedings of the Xth International Conference on Atomic Physics ICAP-X, 1987.

[45] R. Jensen and R. Shankar, *Statistical behavior in deterministic quantum systems with few degrees of freedom*, Phys. Rev. Lett., 54 (1985), pp. 1879–1882.

[46] E. Jessup and D. Sorensen, *A parallel algorithm for computing the singular value decomposition of a matrix*, Technical Report ANL/MCS-TM-102, Argonne National Laboratory, 1987.

[47] ——, *A multiprocessor scheme for the singular value decomposition*, in Parallel Processing for Scientific Computing, G. Rodrigue, ed., SIAM, 1989, pp. 61–66.

[48] N. Johnson and S. Kotz, *Continuous Univariate Distributions*, Houghton Mifflin Company, 1970.

[49] S. Johnsson, *A note on Householder's method, sparse matrices and concurrency*, Memo 4089, Dept. Computer Science, California Institute of Technology, 1980.

[50] S. JOHNSSON, *A computational array for the QR-method*, in Proc. Conference on Advanced Research in VLSI, 1982, P. Penfield, ed., Artech House, Inc., 1982, pp. 123–9.

[51] V. KAHAN, *Accurate eigenvalue of a symmetric tridiagonal matrix*, Technical Report CS41, Dept. Computer Science, Stanford University, 1966 (revised June 1968).

[52] A. KRISHNAKUMAR AND M. MORF, *Eigenvalues of a symmetric tridiagonal matrix: A divide-and-conquer approach*, Numer. Math., 48 (1986), pp. 349–368.

[53] J. KUTTLER AND V. SIGILLITO, *Eigenvalues of the Laplacian in two dimensions*, SIAM Review, 26 (1984), pp. 163–193.

[54] L. LEITHOLD, *The Calculus with Analytic Geometry*, Harper and Row, 1976.

[55] S. LO, B. PHILLIPE, AND A. SAMEH, *A multiprocessor algorithm for the symmetric tridiagonal eigenvalue problem*, SIAM J. Sci. Stat. Comput., 8 (1987), pp. s155–s165.

[56] S. MA, M. PATRICK, AND D. SZYLD, *A parallel, hybrid algorithm for small bandwidth generalized eigenproblems*, Tech. Report CS-1988-28, Department of Computer Science, Duke University, 1988.

[57] ——, *A parallel, hybrid algorithm for the generalized eigenproblem*, in Parallel Processing for Scientific Computing, G. Rodrigue, ed., SIAM, 1989, pp. 82–86.

[58] C. MOLER. Personal Communication, 1987.

[59] B. PARLETT, *The Symmetric Eigenvalue Problem*, Prentice Hall, Englewood Cliffs, NJ, 1980.

[60] J. PETERSON AND A. SILBERSCHATZ, *Operating System Concepts*, Addison-Wesley Publishing Company, 1983.

[61] E. REINGOLD, J. NIEVERGELT, AND N. DEO, *Combinatorial Algorithms*, Prentice Hall, Englewood Cliffs, NJ, 1977.

[62] Y. SAAD, *Shifts of origin for the QR algorithm*, in Proc. of the IFIP Congress, 1974.

[63] Y. SAAD AND M. SCHULTZ, *Some topological properties of the hypercube multiprocessor*, Research Report 389, Dept Computer Science, Yale University, 1984.

[64] ——, *Data communication in hypercubes*, Research Report 428, Dept Computer Science, Yale University, 1985.

[65] A. SAMEH AND D. KUCK, *A parallel QR algorithm for symmetric tridiagonal matrices*, IEEE Trans. Computers, C-26 (1977), pp. 147–53.

[66] R. SCHREIBER, *Bidiagonalization and symmetric tridiagonalization*, Technical Report, Saxpy Computer Corporation, 1985.

[67] D. SCOTT, *Computing a few eigenvalues and eigenvectors of a symmetric band matrix*, SIAM J. Sci. Stat. Comput., 5 (1984), pp. 658–666.

[68] B. SMITH, J. BOYLE, J. DONGARRA, B. GARBOW, Y. IKEBE, V. KLEMA, AND C. MOLER, *Matrix Eigensystem Routines–EISPACK Guide, Lecture Notes in Computer Science, Vol. 6, 2nd edition*, Springer-Verlag, 1976.

[69] J. SPEISER AND H. WHITEHOUSE, *Parallel processing algorithms and architectures for real-time signal processing*, in Proc. SPIE Real Time Signal Processing IV, 1981, pp. 2–9.

[70] M. SPRINGER, *The Algebra of Random Variables*, John Wiley and Sons, 1979.

[71] G. STEWART, *Introduction to Matrix Computations*, Academic Press, New York, 1973.

[72] J. STOER AND R. BULIRSCH, *Introduction to Numerical Analysis*, Springer-Verlag, 1980.

[73] D. SZYLD, *Criteria for combining inverse and Rayleigh quotient iteration*, SIAM J. Num. Anal., 25 (1988), pp. 1369–1375.

[74] D. WATKINS, *Experience with the toda flow method of calculating eigenvalues*, Tech. Report TR-82-1, Dept. of Mathematics, Washington State University, 1982.

[75] J. WILKINSON, *Rounding Errors in Algebraic Processes*, Prentice-Hall, Inc., 1963.

[76] ——, *The Algebraic Eigenvalue Problem*, Clarendon Press, Oxford, 1965.

[77] ——, *Inverse iteration in theory and practice*, Symposia Mathematica Vol.X of the Institute Nationale di Alta Mathematica Monograf, Bologna, 19 (1972), pp. 361–379.

[78] J. WILKINSON AND C. REINSCH, *Handbook for Automatic Computation: Linear Algebra*, Springer-Verlag, 1971.