universität wien

Magisterarbeit

Titel der Magisterarbeit

"Computation of Eigenvectors of Block Tridiagonal Matrices Based on Twisted Factorizations"

Verfasser

Gerhard König

angestrebter akademischer Grad

Diplom-Ingenieur (Dipl.-Ing.)

Wien, 2009

Studienkennzahl laut Studienblatt:	A 066 940
Studienrichtung laut Studienblatt:	Scientific Computing
Betreuer:	Univ.-Ass. Dr. Wilfried Gansterer

Abstract

Computing the eigenvalues and eigenvectors of a band or block tridiagonal matrix is an important aspect of various applications in Scientific Computing. Most existing algorithms for computing eigenvectors of a band matrix rely on a prior tridiagonalization of the matrix. While the eigenvalues and eigenvectors of tridiagonal matrices can be computed very efficiently, the preceding tridiagonalization process can be relatively costly. Moreover, many eigensolvers require additional measures to ensure the orthogonality of the computed eigenvectors, which constitutes a significant computational expense.

In this thesis we explore a new method for computing eigenvectors of block tridiagonal matrices based on twisted factorizations. We describe the basic principles of an algorithm for computing block twisted factorizations of block tridiagonal matrices. We also show some interesting properties of these twisted factorizations and investigate the relation of the block, where the factorizations meet, to an eigenvector of the block tridiagonal matrix. This relation can be exploited to compute the eigenvector in a very efficient way.

Contrary to most conventional techniques, our algorithm for the determination of eigenvectors does not require a reduction of the matrix to tridiagonal form, and attempts to compute a good eigenvector approximation with only a single step of inverse iteration. This idea is based on finding a starting vector for inverse iteration which minimizes the residual of the resulting eigenpair. One of the main contributions of this thesis is the investigation and evaluation of different strategies for the selection of a suitable starting vector.

Furthermore, we present experimental data for the accuracy, orthogonality and runtime behavior of an implementation of the new algorithm, and compare these results with existing methods. Our results show that our new algorithm returns eigenvectors with very low residuals, while being more efficient in terms of computational costs for large matrices and/or for small bandwidths. Due to its structure and inherent parallelization potential, the new algorithm is also well suited for exploiting modern and future hardware, which are characterized by a high degree of concurrency.

Zusammenfassung

Die Berechnung von Eigenwerten und Eigenvektoren von blocktridiagonalen Matrizen und Bandmatrizen stellt einen gewichtigen Aspekt von zahlreichen Anwendungen aus dem Scientific Computing dar. Bisherige Algorithmen zur Bestimmung von Eigenvektoren in solchen Matrizen basierten zumeist auf einer vorhergehenden Tridiagonalisierung der Matrix. Obwohl die Bestimmung von Eigenwerten und Eigenvektoren in tridiagonalen Matrizen sehr effizient durchgeführt werden kann, ist der notwendige Tridiagonalisierungsprozess jedoch sehr rechenintensiv. Des weiteren benötigen zahlreiche Methoden noch Maßnahmen zur Sicherstellung der Orthogonalität der resultierenden Eigenvektoren, was eine zusätzliche Bürde für die Rechenleistung darstellt.

In dieser Arbeit wird eine neue Methode zur Berechnung von Eigenvektoren in blocktridiagonalen Matrizen untersucht, die im Wesentlichen auf der Verwendung von Twisted Factorizations beruht. Hierfür werden die grundlegenden Prinzipien eines Algorithmus zur Berechnung von geblockten Twisted Factorizations von blocktridiagonalen Matrizen erläutert. Des weiteren werden einige interessante Eigenschaften von Twisted Factorizations aufgezeigt, sowie die Beziehung des Blocks, bei dem sich die Faktorisierungen treffen, zu einem Eigenvektor erklärt. Diese Beziehung kann zur effizienten Bestimmung von Eigenvektoren herangezogen werden.

Im Gegensatz zu bisherigen Methoden ist der hier vorgestellte Algorithmus nicht auf eine Reduktion zur tridiagonalen Form angewiesen und beinhaltet nur einen einzigen Schritt der inversen Iteration. Dies wird durch das Auffinden eines Startvektors, der das Residuum des Eigenpaares minimiert, ermöglicht. Einer der Hauptpunkte dieser Arbeit ist daher die Evaluierung verschiedener Strategien zur Selektion eines geeigneten Startvektors.

Des weiteren werden im Rahmen dieser Arbeit Daten zur Genauigkeit, Orthogonalität und des Zeitverhaltens einer Computerimplementation des neuen Algorithmus vorgestellt und mit gängigen Methoden verglichen. Die gewonnenen Daten zeigen nicht nur, daß der Algorithmus Eigenvektoren mit sehr geringen Residuen zurückliefert, sondern auch bei der Berechnung von Eigenvektoren in großen Matrizen und/oder Matrizen mit geringer Bandbreite effizienter ist. Aufgrund seiner Struktur und dem inhärenten Parallelisierungspotential ist der neue Algorithmus

hervorragend dazu geeignet, moderne und zukünftige Hardware auszunutzen, welche von einem hohen Maß an Nebenläufigkeit geprägt sind.

Danksagung

An dieser Stelle möchte ich mich bei allen Personen bedanken, die mich bei der Erstellung dieser Arbeit unterstützt haben.

Besonderer Dank gebührt meinen beiden Betreuern. Einerseits dem Betreuer dieser Diplomarbeit, Prof. Wilfried Gansterer, für die zahlreichen Ideen und Anregungen, durch die ich sehr viel dazugelernt habe, und den großen Zeitaufwand, den er in dieses Projekt gesteckt hat, andererseits dem Betreuer meiner Dissertation, Prof. Stefan Boresch, daß er mir nicht nur immer wieder mit Rat und Tat zur Seite stand, sondern noch dazu die Freiheit gewährt hat, mich so tief in die informatischen Gefilde zu begeben.

Des weiteren möchte ich bei meinen Kollegen bedanken: Den anderen Scientific Computing Studenten, Martin Wimmer und Andreas Gruber, die mich im Rahmen des Studiums vor einigen grauen Haaren bewahrt haben. Außerdem meinen Institutskollegen Christian Schröder, Gregor Neumayr, Michael Haberler, Stefan Bruckner, Sonja Maurer und Thomas Taylor, die mir immer wieder weitergeholfen haben. Spezieller Dank gebührt auch Prof. Othmar Steinhauser für die Erläuterung der etwas obskuren Teile der Literatur.

Schließlich möchte ich mich noch ganz herzlich bei meinen Freunden und vor allem meiner Familie bedanken, die mich stets unterstützt haben, auch wenn sie mich in letzter Zeit nicht mehr so regelmäßig zu Gesicht bekamen.

Contents

1 Introduction **1**

1.1 Motivation . . . 1

1.2 Related Work . . . 3

1.2.1 Synopsis . . . 4

2 Factorizing a Block Tridiagonal Matrix **5**

2.1 Twisted Block LU Factorizations of a Block Tridiagonal Matrix . . . 5

2.1.1 Scalar LU Factorization . . . 5

2.1.2 Block LU Factorization . . . 7

2.1.3 Backward Block LU Factorization . . . 9

2.1.4 Twisted Block LU Factorization . . . 9

3 Inverse Iteration **12**

3.1 Computing an Eigenvector of $W(p)$ . . . 12

3.1.1 Review Inverse Iteration . . . 12

3.1.2 Solving a System of Linear Equations in Inverse Iteration . . . 13

3.1.3 Inverse Iteration Based on Twisted Block Factorizations . . . 15

3.1.4 Connection of the Twisted Factorization to the Inverse of the Matrix . . . 17

3.1.5 Connection of γ to the Eigenvector . . . 23

3.1.6 Choice of Starting Vector . . . 24

4 Results and Discussion **26**

4.1 Experimental Evaluation . . . 26

4.2 Test Data . . . 26

4.3 Comparing Strategies for Selecting the Starting Position . . . 27

4.4 Accuracy of Twisted Block Factorization 31
4.5 Residuals 33
4.6 Orthogonality of the Computed Eigenvectors 41
4.7 Subspaces Identified 47
4.8 Use of Blocked Strategies for the Determination of Eigenvectors Corresponding to Multiple Eigenvalues 49
4.9 Runtimes 50

5 Implementation **55**
5.1 The main program BTEV 55
5.1.1 Matrix data format 56
5.1.2 Workspace requirements 58
5.2 DSYBTEV 58
5.3 DSYBTTWF 59
5.4 DSYBTBS 59
5.5 Auxiliary routines 59
5.6 BLAS/LAPACK routines 60
5.6.1 BLAS 60
5.6.2 LAPACK 61
5.7 Compiling and Usage 63

6 Conclusions **65**
6.1 Conclusions 65
6.2 Future Work 66

Chapter 1

Introduction

In this thesis, we discuss strategies for efficiently computing block factorizations of a block tridiagonal matrix $W(p)$ with p square diagonal blocks. Based on these factorizations, we investigate and empirically evaluate ways for approximating an eigenvector of $W(p)$, given a good approximation of the corresponding eigenvalue.

In the most general setting, $W(p)$ does not have to be symmetric and can be represented in block-wise fashion by a number of submatrices:

$$W(p) := \begin{pmatrix} B_1 & C_1 & & & \\ A_2 & B_2 & C_2 & & \\ & \ddots & \ddots & \ddots & \\ & & A_{p-1} & B_{p-1} & C_{p-1} \\ & & & A_p & B_p \end{pmatrix}. \tag{1.1}$$

The dimensions b_i of the p quadratic diagonal blocks B_i $(i = 1, \ldots, p)$ are in the following called *block sizes* and determine shape and size of the $p-1$ subdiagonal blocks A_i $(i = 2, \ldots, p)$, and of the $p-1$ superdiagonal blocks C_i $(i = 1, \ldots, p-1)$. For eigenvector computations, the focus of this paper is on the special case of symmetric $W(p)$ where $B_i = B_i^\top$ for $i = 1, \ldots, p$, and $C_i = A_{i+1}^\top$ for $i = 1, \ldots, p-1$. In the empirical evaluation, we also consider only equally sized blocks (i.e., all $b_i = b$)

1.1 Motivation

Since the block tridiagonal structure can be considered a generalization of band structures, matrices as defined in Equation (1.1) arise in various situations in Scientific Computing. One example is the solution of boundary-value problems of ordinary differential equations (especially using the so-called finite difference method). In particular, they can also be an intermediate result of a preprocessing step for general dense matrices, for example, of a block tridiagonalization process[1] or of a bandwidth reduction process[2, 3]. Such processes enhance the efficiency of several matrix computations, since banded matrices also allow a significant reduction of computational work and storage, by taking advantage of the structure of zeros around the main diagonals.

For block tridiagonal matrices (and, therefore, also banded matrices) the block tridiagonal divide-and-conquer (BD&C) method [4, 5] allows for efficiently approximating eigenvalues and eigenvectors of symmetric $W(p)$ *without* tridiagonalizing it. However, it turns out that the eigenvector accumulation in the divide-and-conquer process can become the performance limiting factor for increasing accuracy requirements. This motivates efforts in investigating efficient alternatives for computing an eigenvector of a symmetric block tridiagonal matrix, given an approximation of the corresponding eigenvalue.

The idea pursued in this thesis is based on representing $W(p)$ as a product of two block bidiagonal matrices or, equivalently, as a product of three matrices (two block bidiagonals with identities along the diagonal and a block diagonal, which would correspond to the notation of some parts of the literature). This representation allows for a fast and efficient inverse iteration process for computing the desired eigenvector. More specifically, among all possible *twisted block factorizations* of $W(p)$, one factorization is selected as the representation to be used in a single step of the inverse iteration process. This idea is motivated by central components of the MRRR algorithm for computing eigenvectors of a symmetric tridiagonal matrix [6].

In the following chapters, we describe the basic principles of an algorithm for computing block twisted factorizations of $W(p)$. Moreover, we design and empirically evaluate an algorithm for computing eigenvector approximations given approximations of the corresponding eigenvalues on the basis of these block twisted factorizations. A central algorithmic question in this approach is how to select the twisted block factorization and how to choose the starting vector for the inverse iteration process. We motivate and compare several strategies and empirically study their effectiveness in terms of numerical accuracy and in terms of computational performance.

1.2 Related Work

In 1990, Demmel and Kahan showed that the Cholesky factorization of a tridiagonal matrix into two bidiagonals can be used to compute all eigenvalues of a symmetric definite tridiagonal matrix to high accuracy [7], since small relative changes in the bidiagonals cause only small relative changes in the small eigenvalues. Later, it was also shown that most (bidiagonal) LDL^T representations of tridiagonal matrices determine the small eigenvalues to high relative accuracy despite possibly large entries in L or D, if these entries are neutralized by small entries in the eigenvector [8].

This induced the development of very efficient methods for the calculation of eigenvectors in tridiagonal matrices. Based on Fernando's solution to Wilkinson's problem [9] (i. e., the search for the position of the largest entry in the eigenvector), Parlett and Dhillon [10] suggested to use twisted factorizations of tridiagonal matrices to compute a good starting vector (i.e, with a small angle to the true eigenvector) for inverse iteration. This is justified by the fact that the position of the largest component of the eigenvector is associated with the minimal twisted element of the twisted factorizations. They could show that a good starting vector allows for the determination of eigenvectors in a single step of inverse iteration, with the additional benefit that further orthogonalization becomes unnecessary.

While several studies concerning the use of twisted factorizations for the efficient calculation of eigenvectors of *tridiagonal* matrices exist [6, 8, 9, 10, 11, 12, 13], relatively little work has been done on banded or block tridiagonal matrices. Parlett and Dhillon [10] discussed a blocked extension of the tridiagonal case. Their selection of the starting vector for inverse iteration is based on the twisted factorization ($W(p) = LU$) using the subblock of $L_i U_i$ with the minimal singular value (which corresponds to strategy S_{b4} in Section 3.1.6).

Very recently, Vömel and Slemons published a theoretical treatment of twisted factorizations of banded or block-tridiagonal matrices [14]. Therein, they gave a proof of the existence of two twisted factorizations of banded matrices by using a double factorization of the twisted block. Also, the use of twisted factorizations for inverse iteration was mentioned (however, without a practical application or a direct suggestion for the selection of the starting vector) and the connections to the inverse of the matrix were shown.

Thus, experimental data to evaluate the applicability of twisted factorizations in case of block tridiagonal matrices still can not be found in literature. We, therefore, wanted to compare different starting vector selection schemes and the associated advantages and drawbacks of the use of twisted factorizations for inverse iteration.

1.2.1 Synopsis

In Chapter 2, unsymmetric twisted block factorizations of $W(p)$ are discussed. The computation of an eigenvector to a given eigenvalue approximation based on these factorizations is discussed in Chapter 3. Numerical experiments with an implementation of these concepts are summarized in Chapter 4, while details of the implementation of the aforementioned techniques are discussed in Chapter 5. Finally, some conclusions and suggestions for future work are given in Chapter 6.

Chapter 2

Factorizing a Block Tridiagonal Matrix

2.1 Twisted Block LU Factorizations of a Block Tridiagonal Matrix

In analogy to the approach pursued in the MRRR method for tridiagonal matrices[6], we investigate the factorization of block tridiagonal W into two block bidiagonals. LU factorizations of $W(p)$, which yield (unsymmetric) representations of $W(p)$, are discussed in the following sections.

2.1.1 Scalar LU Factorization

In general, a (scalar) LU factorization (or Gaussian Elimination) decomposes a square matrix M into a product of a unit lower triangular matrix L (with only zeros above the diagonal) and an upper triangular matrix U (with only zeros below the diagonal)[15]:

$$M = LU \tag{2.1}$$

Such a factorization is motivated, e.g., by the ease of solving linear systems of equations in triangular matrices (see Section 3.1.2). Also, triangular matrices have various advantageous properties[15]:

- The inverse of a lower/upper triangular matrix is also lower/upper triangular.

- The product of two lower/upper triangular matrices also stays lower/upper triangular.

- The inverse of a *unit* lower/upper triangular matrix (e.g., L in the aforementioned LU factorization) is also unit lower/upper triangular.

- The product of such *unit* lower/upper triangular matrices is also a unit lower/upper triangular matrix.

In case of a general 3×3 matrix, the LU factorization takes the following form:

$$\begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{pmatrix} = \begin{pmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{pmatrix}$$

To create a upper triangular matrix, the subdiagonal elements of the original matrix have to be eliminated. In the standard forward case, this is done by multiplication with elimination matrices L_k whose subdiagonal elements ($i = k+1, ..., n$) of the respective row k equal $l_{ik} = -m_{ik}/m_{kk}$. E.g.:

$$L_1 M = M_1$$

$$\begin{pmatrix} 1 & 0 & 0 \\ -m_{21}/m_{11} & 1 & 0 \\ -m_{31}/m_{11} & 0 & 1 \end{pmatrix} \begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{pmatrix} = \begin{pmatrix} m_{11,1} & m_{12,1} & m_{13,1} \\ 0 & m_{22,1} & m_{23,1} \\ 0 & m_{32,1} & m_{33,1} \end{pmatrix}$$

Note that e.g. m_{22} and $m_{22,1}$ are not the same. In the next step

$$M_2 = L_2 M_1 = L_2(L_1 M)$$

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -m_{32,1}/m_{22,1} & 1 \end{pmatrix} \begin{pmatrix} m_{11,1} & m_{12,1} & m_{13,1} \\ 0 & m_{22,1} & m_{23,1} \\ 0 & m_{32,1} & m_{33,1} \end{pmatrix} = \begin{pmatrix} m_{11,2} & m_{12,2} & m_{13,2} \\ 0 & m_{22,2} & m_{23,2} \\ 0 & 0 & m_{33,2} \end{pmatrix}$$

After n-1 steps, all matrix elements below the main diagonal are eliminated. Thus, M^{n-1} is our sought upper triangular matrix U, while

$$M = L_1^{-1} L_1 M = L_1^{-1} M_1 = L_1^{-1} L_2^{-1} L_2 M_1 = L_1^{-1} L_2^{-1} M_2 = L_1^{-1} L_2^{-1} U$$

which means that $L = L_1^{-1}L_2^{-1}$, or, for a general matrix, the product of all inverses of the matrices: $L = \prod_{k=1}^{n-1} L_k$. It is easy to see that the inverse L_k^{-1} of L_k can be obtained by simply changing the sign of the subdiagonal elements $(\tau^{(k)})$ of L_k, e.g.:

$$\begin{pmatrix} 1 & 0 & 0 \\ l_{21,1} & 1 & 0 \\ l_{21,1} & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ -l_{21,1} & 1 & 0 \\ -l_{21,1} & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ l_{21,1} - l_{21,1} & 1 & 0 \\ l_{21,1} - l_{21,1} & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

If $\tau^{(k)}$ is the vector of the subdiagonal multipliers of L_k, then L is defined by:

$$L = \left(I - \tau^{(1)}e_1^T\right) \cdots \left(I - \tau^{(n-1)}e_{n-1}^T\right) = I - \sum_{k=1}^{n-1} \tau^{(k)}e_k^T \tag{2.2}$$

Thus, L can be formed by simply inserting the elements of the k^{th} column of L_k in the corresponding column of L.

However, problems may arise, if during the factorization process a zero is encountered in the main diagonal element (or, in floating-point arithmetic, stability problems might arise if m_{kk} is very small).

To evade this problem, pivoting strategies have been devised. It is possible to interchange the current row with a subsequent row whose diagonal element is not zero (Usually, the largest element in absolute value of the current column k is selected and the corresponding rows of the submatrix yet to be processed are interchanged in order to make m_{kk} as large as possible.) . Such an action is called (partial) pivoting, and can be represented as a multiplication of the factorization with a pivoting matrix P from the left:

$$M = PLU$$

In the following, we generalize this scalar forward LU factorization process to a block-based LU factorization process for block tridiagonal $W(p)$.

2.1.2 Block LU Factorization

When generalized to a block tridiagonal matrix $W(p)$, the resulting factors L and U will be lower and upper block bidiagonal, respectively. We illustrate the process for $p = 4$. Based on the ansatz

$$
\begin{aligned}
W(4) &= \begin{pmatrix} B_1 & C_1 & & \\ A_2 & B_2 & C_2 & \\ & A_3 & B_3 & C_3 \\ & & A_4 & B_4 \end{pmatrix} \\
&= \begin{pmatrix} P_1 & & & \\ & P_2 & & \\ & & P_3 & \\ & & & P_4 \end{pmatrix} \begin{pmatrix} L_1 & & & \\ M_2 & L_2 & & \\ & M_3 & L_3 & \\ & & M_4 & L_4 \end{pmatrix} \begin{pmatrix} U_1 & N_1 & & \\ & U_2 & N_2 & \\ & & U_3 & N_3 \\ & & & U_4 \end{pmatrix} \qquad (2.3) \\
&= \begin{pmatrix} P_1L_1U_1 & P_1L_1N_1 & & \\ P_2M_2U_1 & P_2L_2U_2 + P_2M_2N_1 & P_2L_2N_2 & \\ & P_3M_3U_2 & P_3L_3U_3 + P_3M_3N_2 & P_3L_3N_3 \\ & & P_4M_4U_3 & P_4L_4U_4 + P_4M_4N_3 \end{pmatrix},
\end{aligned}
$$

the defining equations for the block tridiagonal LU factorization process can be derived block by block. Starting from the top left corner (in "forward" direction), the first step is to factorize $B_1 = P_1L_1U_1$ using partial pivoting. Then, from the equations

$$
\begin{aligned}
P_1L_1N_1 &= C_1 \\
P_2M_2U_1 &= A_2 \qquad (2.4)
\end{aligned}
$$

the matrices N_1 and $M_2' := P_2M_2$ can be computed as solutions of two triangular systems. Note, for arbitrary C_1 and A_2 these linear systems have a unique solution only if B_1 is non singular.

Rewriting the next equation $B_2 = P_2L_2U_2 + M_2'N_1$ into

$$B_2 - M_2'N_1 = P_2L_2U_2 \qquad (2.5)$$

reveals that the next step is to factorize $B_2 - M_2'N_1$ with partial pivoting in order to compute P_2, L_2, and U_2. Note that only at this point P_2 is computed explicitly (so far, it was only contained implicitly in the solution of Equation (2.4)).

Now we can proceed with solving linear systems for N_2 and $M_3' := P_3M_3$, factorizing $B_3 - M_3'N_2$, solving for N_3 and $M_4' := P_4M_4$, and finally factorizing $B_4 - M_4'N_3$. As a result, the entire block LU factorization (2.1.2) of $W(4)$ has been constructed.

2.1.3 Backward Block LU Factorization

The block tridiagonal LU factorization can also be performed in *reverse* direction, starting from the factorization of the lower right block B_p. In this case, the resulting L and U will be upper and lower block bidiagonal, respectively. Again, we illustrate the process for $p = 4$. Based on the ansatz

$$W(4) = \begin{pmatrix} P_1 & & & \\ & P_2 & & \\ & & P_3 & \\ & & & P_4 \end{pmatrix} \begin{pmatrix} L_1 & M_1 & & \\ & L_2 & M_2 & \\ & & L_3 & M_3 \\ & & & L_4 \end{pmatrix} \begin{pmatrix} U_1 & & & \\ N_2 & U_2 & & \\ & N_3 & U_3 & \\ & & N_4 & U_4 \end{pmatrix} \tag{2.6}$$

$$= \begin{pmatrix} P_1L_1U_1 + P_1M_1N_2 & P_1M_1U_2 & & \\ P_2L_2N_2 & P_2L_2U_2 + P_2M_2N_3 & P_2M_2U_3 & \\ & P_3L_3N_3 & P_3L_3U_3 + P_3M_3N_4 & P_3M_3U_4 \\ & & P_4L_4N_4 & P_4L_4U_4 \end{pmatrix}$$

we factorize $B_4 = P_4L_4U_4$ using partial pivoting. From the equations

$$\begin{aligned} P_3M_3U_4 &= C_3 \\ P_4L_4N_4 &= A_4 \end{aligned} \tag{2.7}$$

the matrices N_4 and $M_3' := P_3M_3$ can be computed as solutions of two linear systems, assuming (as before) that B_4 is non singular. Now B_3 can be rewritten as $B_3 = P_3L_3U_3 + M_3'N_4$, which leads to

$$B_3 - M_3'N_4 = P_3L_3U_3.$$

Thus, factorizing $B_3 - M_3'N_4$ with partial pivoting yields P_3, L_3, and U_3. Proceeding analogously to the forward case discussed in Section 2.1.2 determines the remaining unknown submatrices in (2.6).

2.1.4 Twisted Block LU Factorization

Twisted factorizations of $W(p)$ combine $f - 1$ forward elimination steps with $p - f$ backward elimination steps, which we will denote as a $TF(f)$ twisted factorization. Thus, TF(f) denotes a twisted block factorization for $f = 2 \cdots p - 1$, while in the special case $f = p$ it denotes a pure forward factorization and $f = 1$ denotes a pure backward factorization. In the following, we illustrate a block version of a $TF(3)$ twisted block factorization of $W(4)$, where two elimination steps are done in forward direction, and one in backward direction before calculating the block where the two factorizations meet. In order to distinguish the steps done in forward and backward direction, the blocks constructed in the forward direction are marked by the superscript "+", while the blocks constructed in the backward direction are marked by the superscript "−". Note that forward and backward elimination processes are completely independent of each other until the computation of the blocks in the row where the two directions meet (and, therefore, the two factorizations can be parallelized).

Based on the ansatz

$$W(4) = \begin{pmatrix} P_1 & & & \\ & P_2 & & \\ & & P_3 & \\ & & & P_4 \end{pmatrix} \begin{pmatrix} L_1^+ & & & \\ M_2^+ & L_2^+ & & \\ & M_3^+ & L_3 & M_4^- \\ & & & L_4^- \end{pmatrix} \begin{pmatrix} U_1^+ & N_1^+ & & \\ & U_2^+ & N_2^+ & \\ & & U_3 & \\ & & N_3^- & U_4^- \end{pmatrix} \tag{2.8}$$

$$= \begin{pmatrix} P_1L_1^+U_1^+ & P_1L_1^+N_1^+ & & \\ P_2M_2^+U_1^+ & P_2L_2^+U_2^+ + P_2M_2^+N_1^+ & P_2L_2^+N_2^+ & \\ & P_3M_3^+U_2^+ & P_3L_3U_3 + P_3M_3^+N_2^+ + P_3M_4^-N_3^- & P_3M_4^-U_4^- \\ & & P_4L_4^-N_3^- & P_4L_4^-U_4^- \end{pmatrix},$$

we again derive the defining equations block by block.

In analogy to the forward case, the first step is to factorize $B_1 = P_1L_1^+U_1^+$. Then, N_1^+ and $M_2^{'+} := P_2M_2^+$ can be computed as solutions of two linear systems. After updating B_2 as in (2.5) the result is factorized for computing P_2, L_2^+, and U_2^+. Using this information, N_2^+ and $M_3^{'+} := P_3M_3^+$ can be computed as solutions of two linear systems. At this point, the forward part of the TF(3) twisted block factorization is completed, and the next steps are conducted in backward direction. After factorizing $B_4 = P_4L_4^-U_4^-$, N_3^- and $M_4^{'-} := P_3M_4^-$ can be computed as solutions of two linear systems.

Finally, we can work on the third block row where both factorizations meet (we denote this block as "twisted block“). The diagonal block B_3 in this row can be expressed as $B_3 = P_3L_3U_3 + M_3^{'+}N_2^+ + M_4^{'-}N_3^-$. Thus, from factorizing

$$B_3 - M_3^{'+}N_2^+ - M_4^{'-}N_3^- = P_3L_3U_3$$

we finally obtain P_3, L_3, and U_3 and thus have determined all unknown submatrices in (2.8).

For some purposes (for example, when computing eigenvectors of $W(p)$ as discussed in Section 3.1) it is convenient to reformulate the factorization (2.8) as

$$W(4) = LDU$$

with block diagonal D and block tridiagonals L and U which have identity matrices along the diagonal. In particular, for the TF(3) twisted block factorization

$$L = \begin{pmatrix} I & & & \\ M_2^+ \left(L_1^+\right)^{-1} & I & & \\ & M_3^+ \left(L_2^+\right)^{-1} & I & M_4^- \left(L_4^-\right)^{-1} \\ & & & I \end{pmatrix},$$

$$D = \begin{pmatrix} L_1^+ U_1^+ & & & \\ & L_2^+ U_2^+ & & \\ & & L_3 U_3 & \\ & & & L_4^- U_4^- \end{pmatrix},$$

and

$$U = \begin{pmatrix} I & \left(U_1^+\right)^{-1} N_1^+ & & \\ & I & \left(U_2^+\right)^{-1} N_2^+ & \\ & & I & \\ & & \left(U_4^-\right)^{-1} N_3^- & I \end{pmatrix}.$$

Chapter 3

Inverse Iteration

3.1 Computing an Eigenvector of $W(p)$

In this section we discuss how to approximate an eigenvector v of $W(p)$ based on the twisted factorizations of a block tridiagonal matrix as introduced in Section 2.1.4 and assuming that an eigenvalue λ or an approximation $\hat{\lambda}$ thereof is given. The approach pursued is based on one step of inverse iteration on the shifted matrix $W(p) - \lambda I$ based on a suitably chosen twisted factorization of $W(p) - \lambda I$ and a starting vector determined accordingly.

3.1.1 Review Inverse Iteration

An eigenpair $(\lambda,\, v)$ of $W(p)$ satisfies the equation $(W(p) - \lambda I)\, v = 0$. Given an eigenvalue approximation $\hat{\lambda} \approx \lambda$ ($\hat{\lambda}$ will be called "shift" in the following), an approximation $\hat{v}$ for the eigenvector v can be found by inverse iteration.

```
initialize $\hat{v}_{(0)}$, $i := 0$
repeat
    solve $\left(W(p) - \hat{\lambda} I\right) y_{(i+1)} = \hat{v}_{(i)}$
    $\hat{v}_{(i+1)} := y_{(i+1)} / \|y_{(i+1)}\|_2$
    $i := i + 1$
until convergence
```

The starting vector $\hat{v}_{(0)}$ is usually chosen as a random vector [16].

3.1.2 Solving a System of Linear Equations in Inverse Iteration

Solving a linear system $Mx = y$ (of a square matrix M) like in step three of inverse iteration is a central problem of scientific computing[15]. Traditional methods are based on the conversion of the square system to a product of tridiagonal matrices with the LU factorization discussed in Section 2.1.1 ($M = LU$). Due to their structure, solving such systems of equations can be done in a very efficient way, as we show in the next subsections.

Forward Substitution

If we consider the following 3-by-3 lower triangular system:

$$\begin{pmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}$$

The matrix-vector product can be reformulated to:

$$l_{11}x_1 + 0x_2 + 0x_3 = y_1$$

$$l_{21}x_1 + l_{22}x_2 + 0x_3 = y_2$$

$$l_{31}x_1 + l_{32}x_2 + l_{33}x_3 = y_3$$

The unknowns in these equations can be determined in the following fashion: Solving the first equation is straightforward:

$$x_1 = \frac{y_1}{l_{11}}$$

With the result for x_1, we can proceed to the next equation:

$$x_2 = \frac{y_2 - l_{21}x_1}{l_{22}}$$

and finally solve the last equation

$$x_3 = \frac{y_3 - l_{31}x_1 - l_{32}x_2}{l_{33}}$$

This sequential process (shown for an 3-by-3 example) is called *forward substitution.* The general procedure for solving the ith line of $Lx = y$ for x_i is:

$$x_i = \frac{\left(b_i - \sum_{j=1}^{i-1} l_{ij} x_j\right)}{l_{ii}} \tag{3.1}$$

Back Substitution

An analogous procedure can be applied to upper triangular systems:

$$\begin{pmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}$$

However, in case of an upper triangular matrix the simplest equation is located in the last row and the equations are solved in reverse order. This is called back substitution:

$$x_i = \frac{\left(y_i - \sum_{j=i+1}^{n} u_{ij} x_j\right)}{u_{ii}} \tag{3.2}$$

The back substitution algorithm (as well as the forward substitution algorithm) requires n^2 floating point operations.

Using LU Factorizations for Solving a System of Linear Equations

For a given LU factorization of M, we can solve the system $Mx = z$ by transforming it to

$$LUx = z$$

Note that a matrix-vector multiplication yields another vector. If we replace Ux by a yet to be determined vector y, the solution vector x of the system $Mx = z$ can be found by using a combination of two systems of linear equations. We simply perform the forward substitution of the lower triangular matrix L:

$$Ly = z$$

followed by a backward substitution of the upper triangular matrix U

$$Ux = y$$

3.1.3 Inverse Iteration Based on Twisted Block Factorizations

Given a block twisted factorization $W(p) - \hat{\lambda}I = PLU$ as defined in Equation (2.8), we can employ this factorization for solving the system of equations as required in the third step of inverse iteration ($\left(W(p) - \hat{\lambda}I\right) y_{(i+1)} = \hat{v}_{(i)}$). This process is performed (analogous to a normal LU factorization) in three steps:

1. Apply the inverse (P^{-1}) of the pivoting matrix P (whose computation is trivial) to both sides:

   $$LUx = P^{-1}\hat{v}_{(i)} = z$$

   where we denote the resulting vector ($P^{-1}\hat{v}_{(i)}$) on the right side by z.

2. Solve $La = z$ for a via a combined forward and "back" substitution

3. Solve $Uy_{(i+1)} = a$ for $y_{(i+1)}$ via combined back and forward substitution.

Again, we illustrate this for the case $p = 4$ and TF(3). The subvectors ($\vec{a}_i$, $vecz_i$) of length b (corresponding to the matrix blocks) are marked with indices, which correspond to the respective row of blocks (e.g., $\vec{a}_1$):

$$\begin{pmatrix} L_1^+ & & & \\ M_2^+ & L_2^+ & & \\ & M_3^+ & L_3 & M_4^- \\ & & & L_4^- \end{pmatrix} \begin{pmatrix} \vec{a}_1 \\ \vec{a}_2 \\ \vec{a}_3 \\ \vec{a}_4 \end{pmatrix} = \begin{pmatrix} \vec{z}_1 \\ \vec{z}_2 \\ \vec{z}_3 \\ \vec{z}_4 \end{pmatrix}$$

The process of solving a system of equations using a block twisted factorization can be subdivided into multiple block-wise operations. Since both $\vec{a}_2$ and $\vec{a}_4$ have to be known before we can solve the twisted block (i.e., the block where the factorizations meet), it is necessary to start at both ends and gradually solve the equations in inward direction. First, we start by using forward substitution of the forward factorization part (marked with +):

$$L_1^+\vec{a}_1 = \vec{z}_1$$

The solution of the first row ($\vec{a}_1$) can now be used to solve the next row of blocks

$$L_2^+\vec{a}_2 = \vec{z}_2 - M_2^+\vec{a}_1$$

Since the next block is already the block where the factorization meet, we require $\vec{a}_4$ before we can proceed. Thus, we have to solve the equations associated with the backward factorization beforehand (Note that on matrix level, this is actually done by forward substitution, since L_4^- is lower triangular).

$$L_4^- \vec{a}_4 = \vec{z}_4$$

Thus, the row of blocks where the forward and the backward factorization meet can be solved

$$L_3 \vec{a}_3 = \vec{z}_3 - M_2^+ \vec{a}_2 - M_4^- \vec{a}_4$$

A similar procedure has to be applied to the matrix U.

$$\begin{pmatrix} U_1^+ & N_1^+ & & \\ & U_2^+ & N_2^+ & \\ & & U_3 & \\ & & N_3^- & U_4^- \end{pmatrix} \begin{pmatrix} \vec{y}_1 \\ \vec{y}_2 \\ \vec{y}_3 \\ \vec{y}_4 \end{pmatrix} = \begin{pmatrix} \vec{a}_1 \\ \vec{a}_2 \\ \vec{a}_3 \\ \vec{a}_4 \end{pmatrix}$$

However, this time it is necessary to start at the block k where the factorizations meet (in the above example, block number three) and proceed in outward direction, since $\vec{y}_k$ is necessary to solve both the row $k-1$ and $k+1$ (and, therefore, also all other rows). In the familiar $p = 4$, TF(3) example, the back substitution takes the following form. First, we solve

$$U_3 \vec{y}_3 = \vec{a}_3$$

and, correspondingly, in outward direction:

$$U_4^- \vec{y}_4 = \vec{a}_4 - N_3^- \vec{y}_3$$

$$U_2^+ \vec{y}_2 = \vec{a}_2 - N_2^+ \vec{y}_3$$

$$U_1^+ \vec{y}_1 = \vec{a}_1 - N_1^+ \vec{y}_2$$

So far, we have not specified, *which* one of the p possible block twisted factorizations to use in the inverse iteration process. The choice of one of these factorizations is closely related to the starting vector $\hat{v}_{(0)}$ of inverse iteration. In fact, we will utilize the information provided by all the block twisted factorizations of $W(p) - \hat{\lambda} I$ for determining a suitable starting vector $\hat{v}_{(0)}$. The idea which motivates this procedure is that for a properly chosen starting vector a single step of the inverse iteration process should suffice for determining a good approximation of the eigenvector v. (In contrast to standard inverse iteration, which repeats the process of solving the system of equations until convergence)

3.1.4 Connection of the Twisted Factorization to the Inverse of the Matrix

In 1997, Parlett and Dhillon [10] defined a way to compute eigenvectors of tridiagonal matrices using only a single step of inverse iteration given a very accurate approximation to the eigenvalue. For the purpose of determing the optimal starting vector, they were using twisted factorizations, showing the connection of the element where the forward and backward factorization meet with the corresponding diagonal element of the inverse of the matrix and the residual of the resulting approximation to the eigenvector. This approach can be extended to block tridiagonal matrices.

For each possible blocked twisted factorization $TF(k)$, $1 \leq k \leq p$ with blocks of dimension $b \times b$, we define a $b \times b$ block Γ_k (which, therefore, corresponds in its size to the blocks of the twisted factorization) and a $n \times b$ matrix Z (whereof the k^{th} block $Z_k = I$, and the dimensions of Z^+ and Z^- depend on k. Z^+ is a $b(k-1) \times b$ matrix, while Z^- is a $b(p-k) \times b$ matrix). Let there be

$$W(p) \begin{pmatrix} Z^+ \\ I \\ Z^- \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ \Gamma_k \\ \mathbf{0} \end{pmatrix} \tag{3.3}$$

If we use $(W(p) - \lambda I)$ instead of $W(p)$ in the above equation, it becomes clear that Z must be a good approximation to the eigenvector corresponding to λ, if $\|\Gamma_k\|$ is very small. We, therefore, in the following prove that Equation (3.3) exists.

By omitting the k^{th} row of blocks in this equation, there are two remaining homogeneous systems: One based on forward block LU factorization and the other based on backward block LU factorization. In our notation, $U_{1:k-1}$ denotes the whole *submatrix* $(1:k-1, 1:k)$ of U in the LU factorization $W(p) = LU$ (*including* the blocks denoted by M or N):

$$U_{1:k-1} = \begin{pmatrix} U_1^+ & N_1^+ & & & \\ & U_2^+ & N_2^+ & & \\ & & \ddots & \ddots & \\ & & & U_{k-1}^+ & N_{k-1}^+ \end{pmatrix}$$

$$U_{k+1:p} = \begin{pmatrix} N_{k+1}^- & U_{k+1}^- & & & \\ & N_{k+2}^- & U_{k+2}^- & & \\ & & \ddots & \ddots & \\ & & & N_p^- & U_p^- \end{pmatrix}$$

When referring to a particular *block*, it carries only a single index (e.g., M_k). The two (independent) systems in Equation (3.3) are

$$L_{1:k-1}^+ U_{1:k-1}^+ Z^+ = \mathbf{0} \tag{3.4}$$

and

$$L_{k+1:p}^- U_{k+1:p}^- Z^- = \mathbf{0} \tag{3.5}$$

Assuming that the LU factorization exists, the matrices $L_{1:k-1}^+$,$U_{1:k-1}^+$, $L_{k+1:p}^-$ and $U_{k+1:p}^-$ must be invertable. We, therefore, can premultiply Equations (3.4) and (3.5) by the respective inverses to obtain

$$U_{1:k-1}^+ Z^+ = \mathbf{0} \tag{3.6}$$

$$U_{k+1:p}^- Z^- = \mathbf{0} \tag{3.7}$$

Recalling the structure of U in Equation (2.8) (for the example of $W(4)$, $TF(3)$) :

$$\begin{pmatrix} U_1^+ & N_1^+ & & \\ & U_2^+ & N_2^+ & \\ & & U_3 & \\ & & N_3^- & U_4^- \end{pmatrix} \begin{pmatrix} Z_1 \\ Z_2 \\ I \\ Z_4 \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ \mathbf{0} \\ \Gamma_k \\ \mathbf{0} \end{pmatrix}$$

(where $\begin{pmatrix} Z_1 \\ Z_2 \end{pmatrix} = Z^+$ and $Z_4 = Z^-$) the last row of blocks of equations in Equation (3.6) is

$$U_{k-1}^+ Z_{k-1} + N_{k-1}^+ Z_k = \mathbf{0} \tag{3.8}$$

while the first row of blocks of equations in Equation (3.7) is

$$N_{k+1}^- Z_k + U_{k+1}^- Z_{k+1} = \mathbf{0} \tag{3.9}$$

Since $Z_k = I$, we can solve for the respective blocks of Z:

$$Z_{k-1} = -\left(U_{k-1}^+\right)^{-1} N_{k-1}^+ \tag{3.10}$$

$$Z_{k+1} = -\left(U_{k+1}^-\right)^{-1} N_{k+1}^- \tag{3.11}$$

With these solutions we draw our attention again on Equation (3.3)

$$\begin{pmatrix} B_1 & C_1 & & & \\ A_2 & B_2 & C_2 & & \\ & \ddots & \ddots & \ddots & \\ & & & A_p & B_p \end{pmatrix} \cdot \begin{pmatrix} Z^+ \\ I \\ Z^- \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ \Gamma_k \\ \mathbf{0} \end{pmatrix}$$

and select the kth row of blocks. Thus, we obtain the following equation

$$A_k Z_{k-1} + B_k + C_k Z_{k+1} = \Gamma_k \tag{3.12}$$

We can now substitute Z_{k-1} and Z_{k-1} by Equations(3.10) and (3.11)

$$-A_k \left(U_{k-1}^+\right)^{-1} N_{k-1}^+ + B_k - C_k \left(U_{k+1}^-\right)^{-1} N_{k+1}^- = \Gamma_k \tag{3.13}$$

Recalling that according to Section 2.1.4 $A_k = M_k^+ U_{k-1}^+$ and, therefore, $M_k^+ = A_k \left(U_{k-1}^+\right)^{-1}$ and $M_k^- = C_k \left(U_{k+1}^-\right)^{-1}$. We obtain

$$\Gamma_k = -M_k^+ N_{k-1}^+ + B_k - M_k^- N_{k+1}^- \tag{3.14}$$

which, in our notation, according to Section 2.1.4 also means

$$\Gamma_k = L_k U_k \tag{3.15}$$

With the definition of Γ_k we can look at another aspect of Equation (3.3), which can also be written as

$$\begin{pmatrix} Z^+ \\ I \\ Z^- \end{pmatrix} = W(p)^{-1} \begin{pmatrix} \mathbf{0} \\ \Gamma_k \\ \mathbf{0} \end{pmatrix}$$

By looking at the kth row of blocks

$$I = W(p)^{-1}_{(k,k)} \Gamma_k$$

where $W(p)^{-1}_{(k,k)}$ denotes the k^{th} block in the k^{th} row of block of the inverse $(W(p)^{-1})$ of $W(p)$ we can also find a relation of Γ_k to the corresponding diagonal block of the inverse of the matrix $W(p)$

$$\Gamma_k^{-1} = W(p)^{-1}_{(k,k)} \tag{3.16}$$

This means that the inverse of Γ_k is the main diagonal block of the k^{th} row of the inverse of $W(p)$.

To use Equation (3.15) to approximate an eigenvector of $(W(p) - \hat{\lambda} I)$, according to Parlett and Dhillon[10] we have to compute all Γ and find $\hat{\Gamma}_k$ with the minimal singular value. Let

$$\hat{\Gamma}_k v = u \sigma_{min}, \quad \|u\| = \|v\| = 1 \tag{3.17}$$

define a minimal singular triple(σ_{min}, u, v). Then

$$W(p)\,(Z\vec{v}) = \begin{pmatrix} \vec{0} \\ u \\ \vec{0} \end{pmatrix} \sigma_{min} \tag{3.18}$$

If σ_{min} is small enough, then $Z\vec{v}$ is a good initial approximation to an eigenvector of $W(p)$.

An Alternative Derivation

The above block-wise derivation can, in principle, also be extended to a scalar choice of the starting position (and, therefore, the starting vector for inverse iteration). Together with the relation in Section 3.1.5, we can use twisted block LU factorizations to evaluate the quality of each possible position of the starting vector. For this purpose, we use a vector $\vec{z}$ of length n, which can be subdivided into p smaller vectors of length b (instead of the $n \times b$ matrix Z of the above block-wise derivation):

$$\vec{z} = \begin{pmatrix} \vec{z}_1 \\ \vec{z}_2 \\ \vdots \\ \vec{z}_p \end{pmatrix}, \quad \vec{z}(s) = 1, \quad (k-1)b + 1 \leq s \leq kb$$

On position s, the vector $\vec{z}$ equals 1 and s is located in the subvector $\vec{z}_k$ (which corresponds in its position to the k^{th} row of blocks). This establishes the relation to a single position of the eigenvector (by omitting the equation corresponding the respective position of the eigenvector). Let there be

$$W(p)\vec{z} = \vec{e}_s \gamma_s \tag{3.19}$$

Where $W(p)$ is our block tridiagonal matrix and $\vec{e}_s$ is a vector which, except for the s^{th} entry (located in the k^{th} row of blocks) contains zeros ($e_s(s) = 1$, $\|\vec{z}\| = 1$) If we omit the k^{th} row of blocks of this equation, there are again two remaining homogeneous systems (one corresponding to the forward factorization part, the other corresponding to the backward factorization). We thus obtain

$$L^+_{1:k-1} U^+_{1:k-1} \vec{z}_{1:k-1} = \vec{0} \tag{3.20}$$

$$L^-_{k+1:p} U^-_{k+1:p} \vec{z}_{k+1:p} = \vec{0} \tag{3.21}$$

We premultiply Equations (3.20) and (3.21) by the respective inverses to obtain

$$U^+_{1:k-1} \vec{z}_{1:k-1} = \vec{0} \tag{3.22}$$

$$U^-_{k+1:p} \vec{z}_{k+1:p} = \vec{0} \tag{3.23}$$

Now the last row of blocks of equations in Equation (3.22) is

$$U^{+}_{k-1}\vec{z}_{k-1} + N^{+}_{k-1}\vec{z}_{k} = \mathbf{0} \tag{3.24}$$

while the first row of blocks of equations in Equation (3.23) is

$$N^{-}_{k+1}\vec{z}_{k} + U^{-}_{k+1}\vec{z}_{k+1} = \mathbf{0} \tag{3.25}$$

We now solve for the respective subvectors of $\vec{z}$:

$$\vec{z}_{k-1} = -\left(U^{+}_{k-1}\right)^{-1} N^{+}_{k-1}\vec{z}_{k} \tag{3.26}$$

$$\vec{z}_{k+1} = -\left(U^{-}_{k+1}\right)^{-1} N^{-}_{k+1}\vec{z}_{k} \tag{3.27}$$

The kth row of blocks in Equation (3.19) is

$$A_k\vec{z}_{k-1} + B_k\vec{z}_k + C_k\vec{z}_{k+1} = e_s((k-1)b+1:kb)\gamma_s \tag{3.28}$$

where we use Equations (3.24) and (3.25) to find

$$-A_k\left(U^{+}_{k-1}\right)^{-1} N^{+}_{k-1}\vec{z}_k + B_k\vec{z}_k - C_k\left(U^{-}_{k+1}\right)^{-1} N^{-}_{k+1}\vec{z}_k = \vec{e}_s((k-1)b+1:kb)\gamma_s \tag{3.29}$$

The left hand side of Equation (3.29) we can substitute by

$$-A_k\left(U^{+}_{k-1}\right)^{-1} N^{+}_{k-1}\vec{z}_k + B_k\vec{z}_k - C_k\left(U^{-}_{k+1}\right)^{-1} N^{-}_{k+1}\vec{z}_k = \Gamma_k\vec{z}_k \tag{3.30}$$

where Γ_k is the same as in Equation (3.30). We thus obtain

$$\Gamma_k\vec{z}_k = \vec{e}_s((k-1)b+1:kb)\gamma_s \tag{3.31}$$

By premultiplying both sides with Γ_k^{-1} and looking at the s^{th} equation only we can see that

$$\gamma_s^{-1} = [\Gamma_k^{-1}]_{s,s} \tag{3.32}$$

which, according to Equation (3.16), is also related to the inverse of $W(p)$.

However, the calculation of the inverse of $W(p)$ is not a viable procedure to determine γ_s. We therefore reformulate Equation (3.31) to find

$$\Gamma_k \frac{\vec{z}}{\gamma_s} = \vec{e}_s$$

If $\vec{g} = \frac{\vec{z}}{\gamma_s}$ we can solve the equation

$$\Gamma_k \vec{g} = L_k U_k \vec{g} = e_s \tag{3.33}$$

for each position s in each block twisted factorization k. Thus, we can calculate γ_s with

$$\gamma_s = \frac{\vec{z}(s)}{\vec{g}(s)} = \frac{1}{\vec{g}(s)} \tag{3.34}$$

The additional computational costs for this procedure would amount to only $2nb^2$ floating point operations (for all backward and forward substitutions only in order to calculate $\vec{z}(s)$ from $L_k U_k z_s = e_s$), since the LU-factorizations necessary for solving the system of equations of each Γ_k were already computed for the twisted factorization. Thus, it should be possible to find the position of the largest entry of the eigenvector.

3.1.5 Connection of γ to the Eigenvector

For the computation of an eigenvector v we need to determine all γ_s (meaning, for each possible position in the eigenvector we have to evaluate its suitability as a starting position for inverse iteration) where

$$\left(W(p) - \lambda I\right) \vec{z}_s = \vec{e}_s \gamma_s \qquad , \vec{z}_s(s) = 1, \quad s = 1, \cdots, n \tag{3.35}$$

If the shift λ is a good approximation to a true eigenvalue, for any s,

$$\left\| \left(W(p) - \lambda I\right) \vec{z}_s \right\| = |\gamma_s| \tag{3.36}$$

Thus, by determining the position s with the minimal γ_s, we minimize the residual associated with the computed eigenvector z_s.

In the next section, we will empirically evaluate different approaches to select the starting vector based on γ_s and Γ_k

3.1.6 Choice of Starting Vector

In the following, we motivate and specify several strategies for determining the starting vector $\hat{v}_{(0)}$ for the inverse iteration process on $W(p) - \hat{\lambda} I$. In Section 4.1, these strategies are compared experimentally in terms of the resulting quality of the eigenvector approximation if only *one step* of inverse iteration is performed.

Generally, we restrict ourselves to starting vectors $\hat{v}_{(0)}$ with an element of value one in position j and zeros in all other positions (except in reference implementation S_r). When solving a block bidiagonal system with such a vector $\hat{v}_{(0)}$ as the right hand side, all entries of the solution vector below position j will be zero. Thus, in the following we will call the position j "starting position" of the back- or forward substitution process and often identify this starting position with the starting vector $\hat{v}_{(0)}$ (since j completely determines $\hat{v}_{(0)}$).

Scalar Strategies

- **Strategy** S_r: As a reference strategy, we picked a starting vector with random entries between -1 and $+1$ at each position, which corresponds to standard inverse iteration.

- **Strategy** S_s: For tridiagonal matrices, there is a correlation between the components of the eigenvector and the corresponding diagonal elements of the upper bidiagonal matrix U of the LU factorization [10]. Consequently, one way for picking the starting vector $\hat{v}_0$ is to derive it from the position of the diagonal element of U with the minimum absolute value over all possible block twisted factorizations.

  In block tridiagonal matrices this strategy is motivated by Equations 3.33 and 3.34. If $\vec{g}(s)$ is very large then γ_s (and, consequently, the residual of the approximation to the eigenvector) will be small. Now according to Equation 3.2 $\vec{g}(s)$ is probably large, if U_{ss} is very small, i.e., if $U_{ss} << \left(\vec{c}_i - \sum_{j=/s+1}^{n} U_{ij} \vec{g}_j \right)$, where ,in this case, $\vec{c}$ denotes the solution to $L_k \vec{c} = \vec{e}_s$. (Note that since the determinant of a triangular matrix is given by the product of its main diagonal entries, there is also a relation of the main diagonal entries to the eigenvalues of the block. Thus, there might also be a relation to block strategy S_{b4})

If $|U_{mm}|$ is minimum over all diagonal entries of the factors U of all possible twisted factorizations, then $\hat{v}_0$ is defined as a vector of zero entries except for position m, where the entry is one, and the factorization which contains this minimum diagonal element $|U_{mm}|$ is used for solving the linear system.

Block Strategies

In addition to scalar strategies, it seems important to investigate block-oriented strategies for block tridiagonal matrices. Most block strategies are motivated by Equation 3.3. If a suitable norm of Γ_k is small, then the residual of the approximation to the eigenvector should also be small.

Identifying a starting *block* instead of a scalar starting position in principle allows for determining b_i different scalar starting positions and thus potentially for approximating b_i different eigenvectors for an eigenvalue with multiplicity higher than one. The basic idea is to use a $n \times b$ matrix of starting vectors which has a $b \times b$ matrix with entries of value one along the main diagonal in the rows corresponding to the starting block. The strategies defined in the following only differ in the heuristic for determining the starting block m.

- **Strategy** S_{b1}: m is the block number for which the infinity norm (i.e., the largest sum of the entries of a row : $||A||_\infty = \max_i \sum_{j=1}^{n} |a_{ij}|$) of $||L_iU_i||_\infty$ is minimum over all diagonal blocks over all possible twisted factorizations LU.

- **Strategy** S_{b2}: m is the block number for which $\|L_iU_i\|_\infty / \|B_i - \hat{\lambda}I\|_\infty$ is minimum over all diagonal blocks B over all possible twisted factorizations LU (B_i is the corresponding diagonal block in the original matrix $W(p)$).

- **Strategy** S_{b3}: m is the block number for which $\|L_iU_i\|_\infty / \|L_{i+1}U_{i+1}\|_\infty$ is maximum over all main diagonal blocks over all possible twisted factorizations LU (strongest decrease in the norm of diagonal blocks).

- **Strategy** S_{b4}: m is the number of the block which has the smallest singular value over all diagonal blocks L_iU_i over all possible twisted factorizations [10].

Chapter 4

Results and Discussion

4.1 Experimental Evaluation

In this section, we summarize extensive experimental evaluations of the concepts outlined in Section 3.1. After a discussion of the test data used, the accuracy of the twisted block factorization process (as outlined in Section 2.1), the residuals of the computed eigenvectors, their orthogonality, and the subspaces spanned by subsets of the computed eigenvectors are investigated. We also discuss the use of block strategies for solving eigenvectors for multiple eigenvalues. Finally, the runtime performance of the proposed method is evaluated and compared with competing approaches.

4.2 Test Data

Seven different types of symmetric block tridiagonal matrices were generated and used for testing purposes. The types denoted by A1 to A6 are characterized by a certain distribution pattern of their eigenvalues and were generated using software written by Y. Bai.

- **Type R** matrices contain random entries in $[0, 1]$.
- **Type A1** matrices have eigenvalues which are clustered around $\pm\varepsilon_{mach}$.
- **Type A2** matrices have eigenvalues which are clustered around ± 1.

- **Type A3** matrices have eigenvalues which are geometrically distributed from ±1 to ε_{mach}.
- **Type A4** matrices have eigenvalues which are arithmetically distributed from ±1 to ε_{mach}.
- **Type A5** matrices have eigenvalues whose logarithms are uniformly distributed from 1 to $\pm\varepsilon_{mach}$.
- **Type A6** matrices have eigenvalues which are random and uniformly distributed in $[-1, 1]$.

The eigenvalue distributions for concrete test matrices of dimension $n = 500$ are depicted in Figure 4.1. (Matrices of Type R are not depicted in Figure 4.1. Their eigenvalues were distributed between -4 and 9, with the majority of eigenvalues lying between -2 and 2. The minimal absolute gap between two eigenvalues in R was $5.0 \cdot 10^{-5}$).

While most matrix types (R, A3, A4, A5 and A6) are good test systems inasmuch as they represent most normal matrices, matrix types A1 and A2 prove to be especially difficult test cases due to the tight clustering of eigenvalues.

4.3 Comparing Strategies for Selecting the Starting Position

In order to determine the best strategy for the selection of the starting position for the back substitution, 20 eigenvalues (every 25^{th} eigenvalue in increasing order) were selected from matrices of dimension $n = 500$ for all matrix types introduced in Section 4.2 and the six different strategies for determining the starting vector $\hat{v}_0$ introduced in Section 3.1.6 were compared. The different strategies were evaluated based on two criteria: First, the residual resulting from the selected start vector (e_s). Second, the percentage of correctly computed eigenvectors: an eigenvector was considered to be correct if it pointed in the same direction as the corresponding eigenvector computed with `LAPACK/dsyevd`. This was considered the case if the scalar product between the two vectors (i.e., once obtained with twisted factorizations and once by normal means) was greater than 0.99.

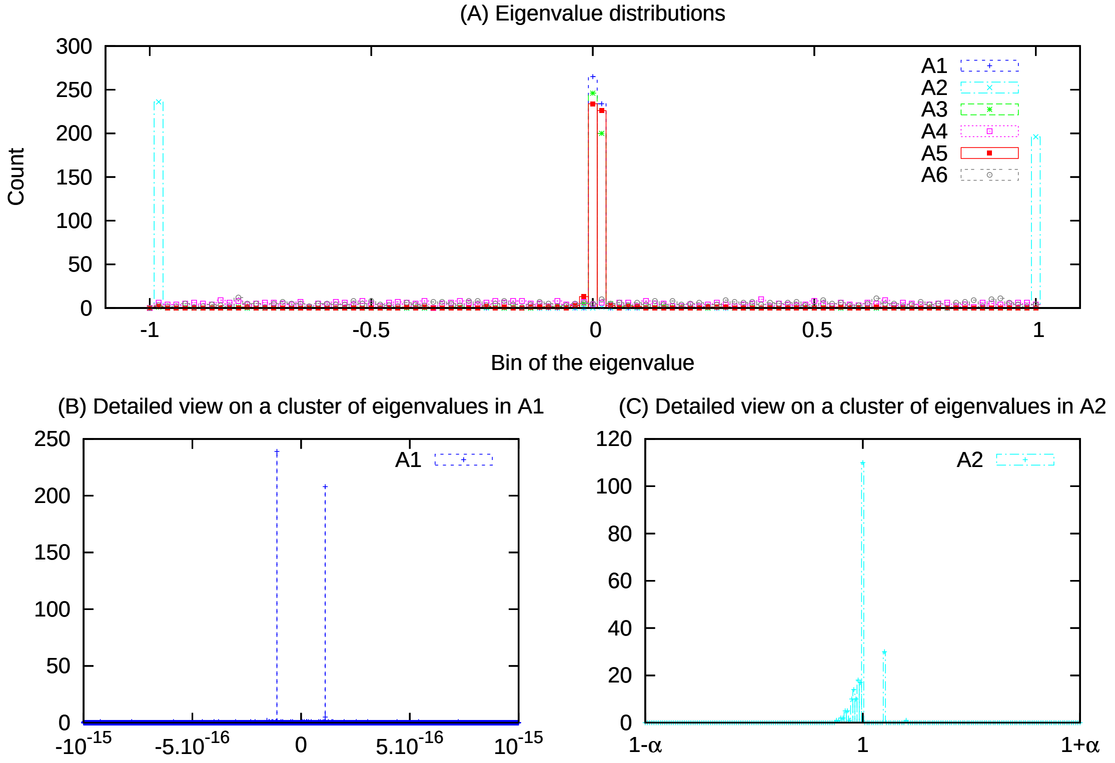

Figure 4.1: (A) Eigenvalue distributions of all test matrix types with a bin width of 0.02. (B) depicts the two very tight clusters of a Type A1 matrix around $\pm\varepsilon_{mach}$ with a bin width of $2.0 \cdot 10^{-19}$, while (C) illustrates the tight cluster of eigenvalues around +1 of a Type A2 matrix with a bin width of 10^{-15} and $\alpha = 10^{-13}$. As in illustration (A), the x-axis shows the value of the eigenvalue, while axis y denotes the number of eigenvalues within each bin.

Table 4.1 summarizes the comparisons of the different strategies for determining the starting vector $\hat{v}_0$. Column "$\bar{\mathfrak{R}}$" lists the average residual (i.e., $\bar{\mathfrak{R}} = \|(W(p) - \lambda I)\hat{v}\|$) over all 140 eigenvector calculations for all matrix types. Column "**overall**" lists the average percentage of correctly computed eigenvectors considering all 140 eigenvector calculations of all matrix types. Columns "**R-5**" to "**A6**" list the average percentage of correctly computed eigenvectors over 20 eigenvector calculations using the respective matrix type for the respective strategy for selecting the starting position (e.g., one correctly predicted eigenvector out of 20 would yield a result of 5%). The first six rows in Table 4.1 represent the strategies introduced in Section 3.1.6, whereas the last two rows show the results achieved with two theoretical reference strategies which are not applicable in practice: row "$\mathbf{S}_{\text{minres}}$" corresponds to selecting the starting vector (e_s) which yields the smallest residual over all possible starting positions (s) in all block twisted factorizations. Therefore, it represents the highest theoretical accuracy which can be achieved by using a starting vector for inverse iteration with all zeros except for the position s, where $e_s(s) = 1$ and $\|e_s\| = 1$. Row "$\mathbf{S}_{\text{optevec}}$", on the other hand, corresponds to the starting position which yields the largest scalar product (best agreement) with the corresponding eigenvector computed using `LAPACK/dsyevd` over all possible starting positions in all block twisted factorizations. This approach is necessary, since in cases of very tight clusters of eigenvalues (i.e., with gaps close to machine precision ε_{mach}) it is possible to achieve a good residual by calculating the eigenvector of a neighboring eigenvalue instead of the true eigenvector, which corresponds to the eigenvalue closest to the shift (i.e., in these cases the residual is not a reliable indicator for the quality of an eigenpair).

Both the scalar strategy S_s (based on the position s of the minimal diagonal element in all U) and the strategy S_{b4} (based on the block with the minimal absolute eigenvalue) consistently yield the best residuals and percentages of correctly computed eigenvectors, and their performance is very close to that of the theoretical reference strategies. Compared to the reference implementation S_r (which is based on a random vector with entries in each element between -1 and $+1$.) both strategies S_s and S_{b4} show a superior residual, surpassing the mean residual of S_r by several orders of magnitude. In the light of these results, we can conclude that

Table 4.1: Comparison of five different strategies for selecting the starting position for back substitution for seven different matrix types over 20 eigenvalue calculations in terms of resulting residual and percentage of correctly computed eigenvectors. All test matrices had dimension $n = 500$ and block size $b = 5$.

Strategy	$\bar{\mathfrak{R}}$	**correctly computed eigenvectors** [%]							
		overall	**R-5**	**A1**	**A2**	**A3**	**A4**	**A5**	**A6**
$\mathbf{S_r}$	$1.89 \cdot 10^{-11}$	69	100	5	0	90	100	90	100
$\mathbf{S_s}$	$1.53 \cdot 10^{-15}$	72	100	5	10	90	100	95	100
$\mathbf{S_{b1}}$	0.10	25	95	0	0	10	50	5	15
$\mathbf{S_{b2}}$	0.02	48	100	0	0	75	100	15	45
$\mathbf{S_{b3}}$	0.06	37	100	5	0	25	70	25	35
$\mathbf{S_{b4}}$	$1.77 \cdot 10^{-14}$	70	100	5	0	90	100	95	100
$\mathbf{S}_{\text{minres}}$	$9.10 \cdot 10^{-16}$	67	100	5	5	90	100	95	100
$\mathbf{S}_{\text{optevec}}$	$7.57 \cdot 10^{-12}$	74	100	10	20	90	100	100	100

employing blocked twisted factorizations can significantly improve the performance of the first step of inverse iteration.

On the other hand, all block strategies, with the noteable exception of strategy S_{b4}, were unable to give consistently satisfactory results. While all strategies seem to be effective in matrix type R-5, most block strategies fail in matrix types A1, A2, A5 and A6. Selection scheme S_{b2} seems to be more or less successful in matrix types A3 and A4, but also fails in A5 and A6, so its overall performace is far from adequate. Strategies S_{b1} and S_{b3} do not yield acceptable results in any matrix type except matrix type R-5. We correspondingly conclude that the only viable block strategy is S_{b4}, which is the computationally most costly strategy, depending on the absolute values of the singular values of the block, contrary to simple matrix norms (like, e.g., the infinity norm $\|X\|_\infty$) in the cases $S_{b1} - S_{b3}$.

As expected, Table 4.1 also illustrates that in cases with tightly clustered eigenvalues (test matrices A1 and A2) there are many instances where no starting position yields accurate eigenvectors with a single step of inverse iteration (compared to `LAPACK/dsyevd`, see row "$\mathbf{S}_{\text{optevec}}$), even though the residuals in A1 and A2 are also very small (data not shown). This demonstrates that the effectiveness of starting vectors based on a single entry (as well as random starting vectors as in case of S_r) is limited. While it possible to improve the results for the first step of inverse iteration, there seem to be cases where a single step of inverse iteration is not enough to satisfy a low residual *and* orthogonality of the eigenvectors. Other test matrices, e.g., matrix type R-5 (which is filled with random entries) lead to good results irrespective of the strategy for the selection of the starting vector for inverse iteration. These matrix types are, therefore, unable to discriminate between effective and inadequate selection schemes for the starting vector.

As mentioned before, an eigenvector was considered computed correctly if the scalar product with the corresponding eigenvector as obtained from `LAPACK/dsyevd` was larger than 0.99. Obviously, the choice of this threshold influences the percentage of correctly computed eigenvectors. If a more stringent criterion for "correctness" is applied (a threshold for the scalar product of 0.999999 instead of 0.99), the overall percentage of correctly computed eigenvectors over all matrix types using the strategies $S_{optevec}$, S_s and S_{b4} become almost equal (dropping to approximately 61.67%). This indicates that these two selection strategies yield the most accurate results possible with a single step of inverse iteration.

We conclude that the strategies S_s and S_{b4} yield in general the best results. Whereas S_s is slightly cheaper than S_{b4} (the difference depends on the block size), it does not provide a guideline how to compute a basis for a subspace corresponding to an eigenvalue with multiplicity greater than one. In the following, we will focus on strategy S_s, except for Section 4.8, where we specifically consider higher multiplicities of eigenvalues and discuss more experiments with strategy S_{b4}.

4.4 Accuracy of Twisted Block Factorization

The procedure outlined in Section 2.1.4 for computing the twisted LU factorization of $W(p)$ has been implemented in the Fortran routine DSYBTTWF. To test the numeric reliability of such factorizations, multiple tests were conducted. Figure 4.2 depicts the common logarithms of the mean and standard deviation of the factorization error $\|LU - W(p)\|_\infty$ of factorizations conducted for fifty different shifts in the respective matrix type based on selection strategy S_s (see Section 3.1.6). All matrices used were of dimension $n = 500$, and the shifts used correspond to every tenth eigenvalue.

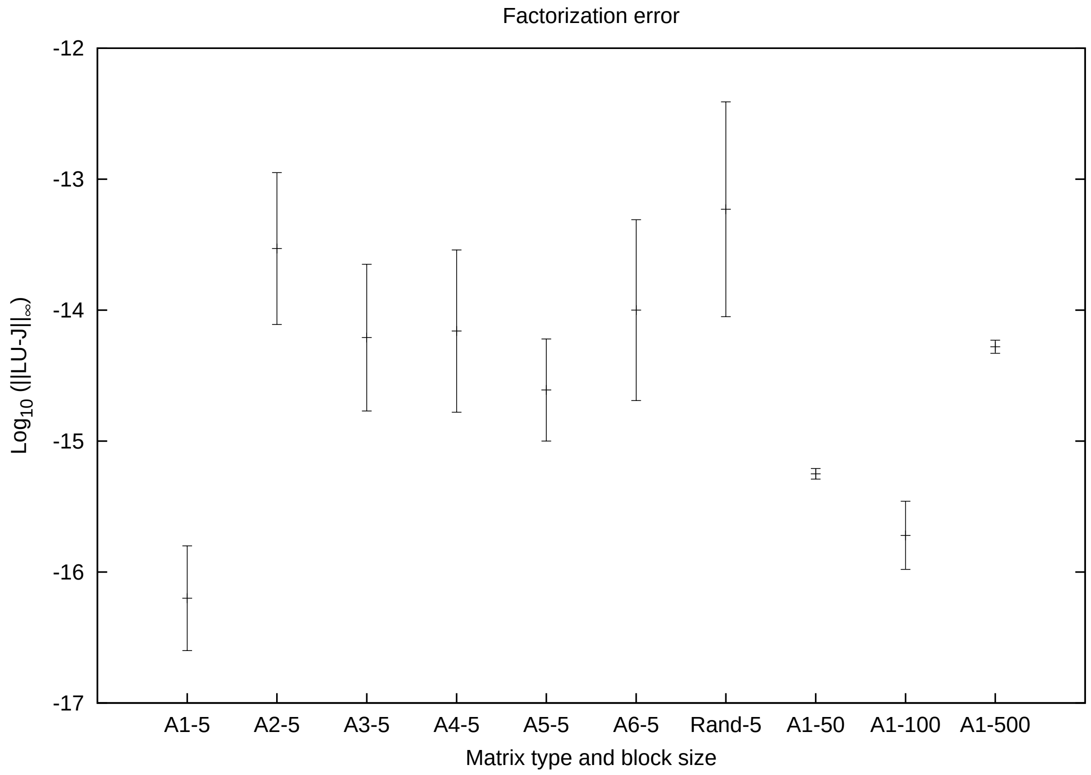

Figure 4.2: Variation of the factorization error $\|LU - W(p)\|_\infty$ over fifty different shifts for different matrix types and block sizes. The first part of the labels on the x-axis denotes the matrix type (e.g., “A1” or “R”), and the second part after the hyphen denotes the block size b.

With the notable exception of matrix type A1 (which generally contains smaller entries than the other matrices), all factorization errors have approximately the same order of magnitude. The mean factorization error grows slightly with increasing block size, as can be seen on the right side of Figure 4.2 for increasing block sizes of random matrices by drawing an imaginary regression line through A1-5, A1-50 and A1-500, while considering A1-100 to be an outlier. (Which is not surprising since more floating point operations are involved in matrices with increasing block sizes.). Therefore, we conclude that the numerical accuracy of the block twisted factorization (which is generally quite high) will not significantly affect the final result of inverse iteration.

Furthermore, we tested whether the quality of the blocked twisted factorization is affected by selecting a shift within a cluster of eigenvalues or by choosing an isolated eigenvalue. For this purpose, we calculated all twisted factorizations for the smallest eigenvalue of matrix type A1, which is an isolated eigenvalue at -1 and for eigenvalue number 301 of matrix type A1, which is located in a cluster of eigenvalues around $+\varepsilon_{mach}$, to determine the factorization errors.: While using an isolated eigenvalue as shift resulted in an average factorization error of $4.26 \cdot 10^{-17}$ $\pm 9.71 \cdot 10^{-18}$, a shift from a cluster of eigenvalues resulted in an error of $4.21 \cdot 10^{-17}$ $\pm 4.86 \cdot 10^{-18}$. We, therefore, conclude that the selection of the shift does not have a great impact on the quality of the twisted factorization.

4.5 Residuals

To verify the overall performance of the starting position prediction method S_s, Figure 4.3 shows the distribution of the residuals of 9000 eigenvector calculations. For data generation, all eigenvalues of all six matrix types with dimensions 500 and 1000 (thus in total 1500 data points per matrix type) were used. In Figure 4.3, the frequency of the corresponding residual is plotted versus the common logarithm of the residual. The residual ranges between $1.2 \cdot 10^{-12}$ and $2.4 \cdot 10^{-34}$. Overall, the mean residual is $2.2 \cdot 10^{-19}$, while the median is at $1.4 \cdot 10^{-16}$ (which is slightly higher than $\varepsilon_{mach} = 1.1 \cdot 10^{-16}$), a result which is quite satisfactory, since the largest peak of the gaussian-like distribution lies below $n\varepsilon_{mach}$. The second peak at about $1.0 \cdot 10^{-32}$ can be mainly attributed to the distribution of shifts in matrix type A1 (Obviously, the matrix entries in A1 are generally smaller than in other matrix types. Therefore, the associated residuals are shifted to left on this plot). Since in normal inverse iteration residuals up to $\sqrt{\varepsilon_{mach}}$ can be expected (which, in our case, would amount to $1.1 \cdot 10^{-8}$), the shown data demonstrates that all eigenvectors calculated with block twisted factorizations and starting vector selection strategy S_s surpass this threshold by several orders of magnitude, leading to good residuals.

Also, the dependence of the residual on the quality of the shift was studied, since the accuracy of the shift is a key factor for the algorithm. For this purpose, we selected a type A6 matrix of dimension 500×500 and calculated all 500 eigenvalues

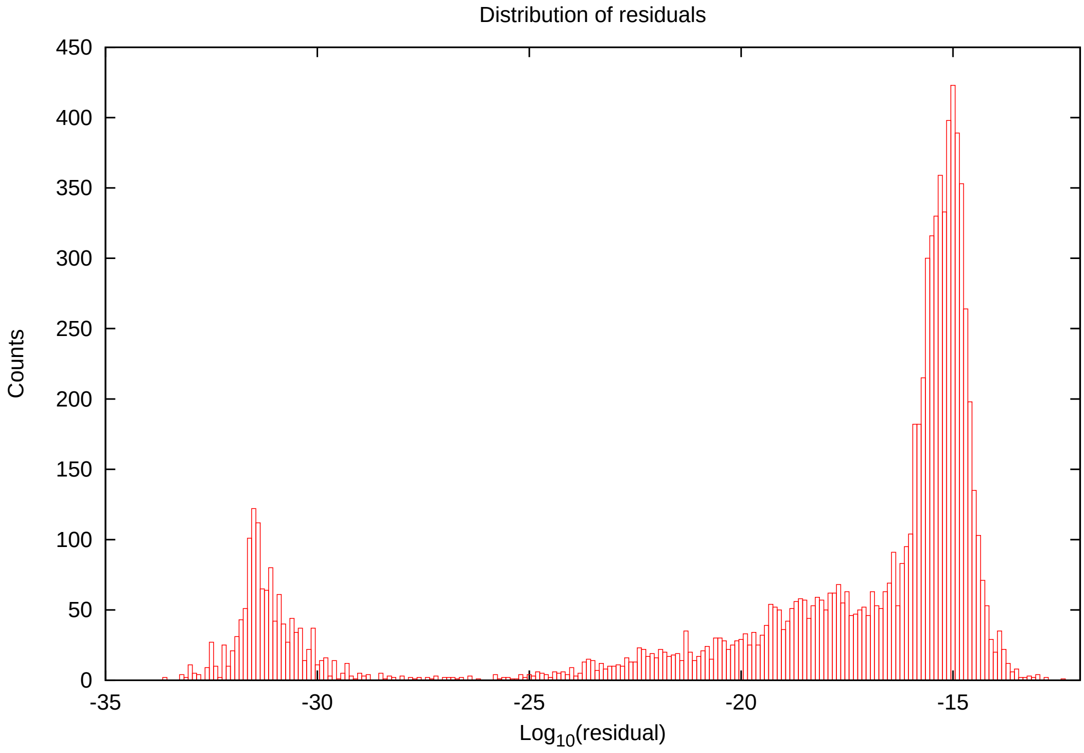

Figure 4.3: Distribution of the residuals of 9000 eigenvector calculations.

of this matrix. The shift were set to equal to all eigenvalues, the corresponding eigenvectors were calculated and the mean residuals and their standard deviations determined. In the next step, each shift was perturbed by a certain deviation (once in positive, once in negative direction) and, again, the mean residuals and their standard deviations were determined. This was done for multiple orders of magnitude of the shift perturbation. The results of this experiment are shown in Figure 4.4.

In Figure 4.4, the common logarithm of the perturbation of the shift is plotted versus the corresponding mean residuals and their standard deviations. As shown in this figure, a linear relationship between the perturbation of the shift and the resulting residual exits. Thus, (at least in the case of matrix type A6) the quality of the resulting eigenvector, as far as the residual is concerned, does not heavily depend on the accuracy of the shift (i.e., the algorithm is quite stable as far as this property is concerned, since the dependence is not quadratic or otherwise nonlinear). Based on this data, it would be admissible to implement the algorithm also in other regimes of precision (e.g, single precision) without suffering disproportionate penalties on accuracy. Also we can see that the curve of the residuals starts to flatten

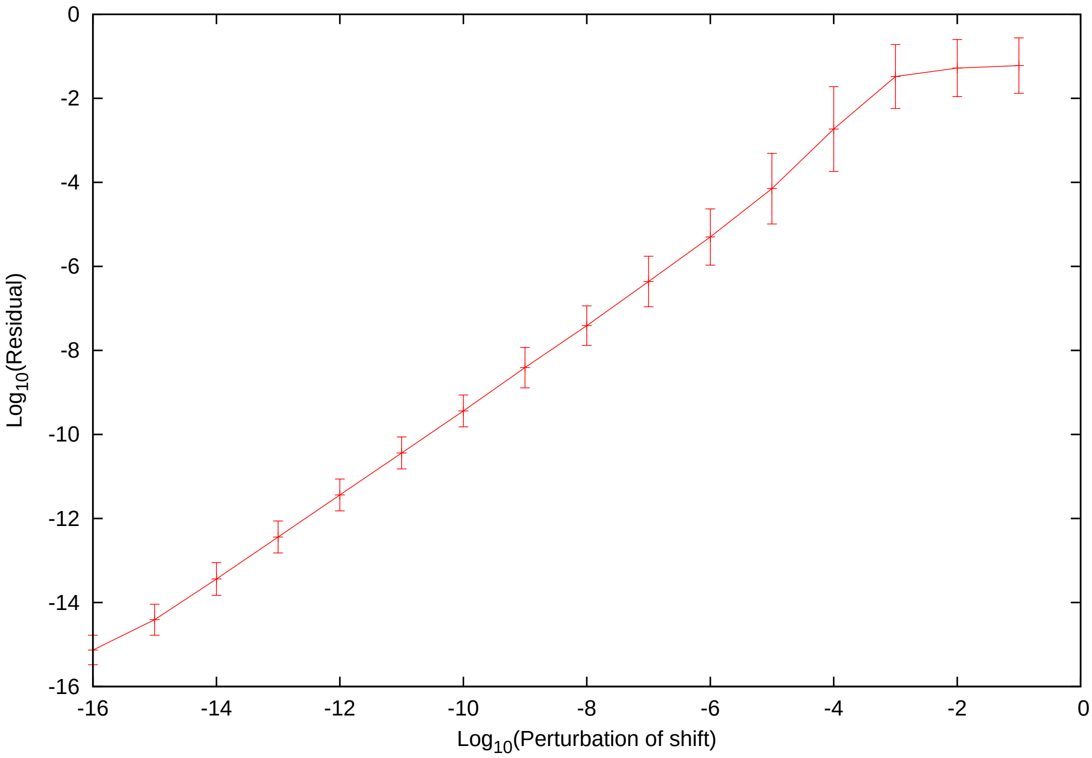

Figure 4.4: Dependence of the residual on the quality of the shift. Using a 500 × 500 matrix of type A6 the mean residuals and the corresponding standard deviations were calculated for 500 different shifts. Each shift was perturbed by adding errors of different magnitude in both positive and negative direction. Thus, each data point is a mean of 1000 residuals.

after perturbations higher than 0.001, suggesting, on the other hand, that there is a minimal acceptable precision of the eigenvalue.

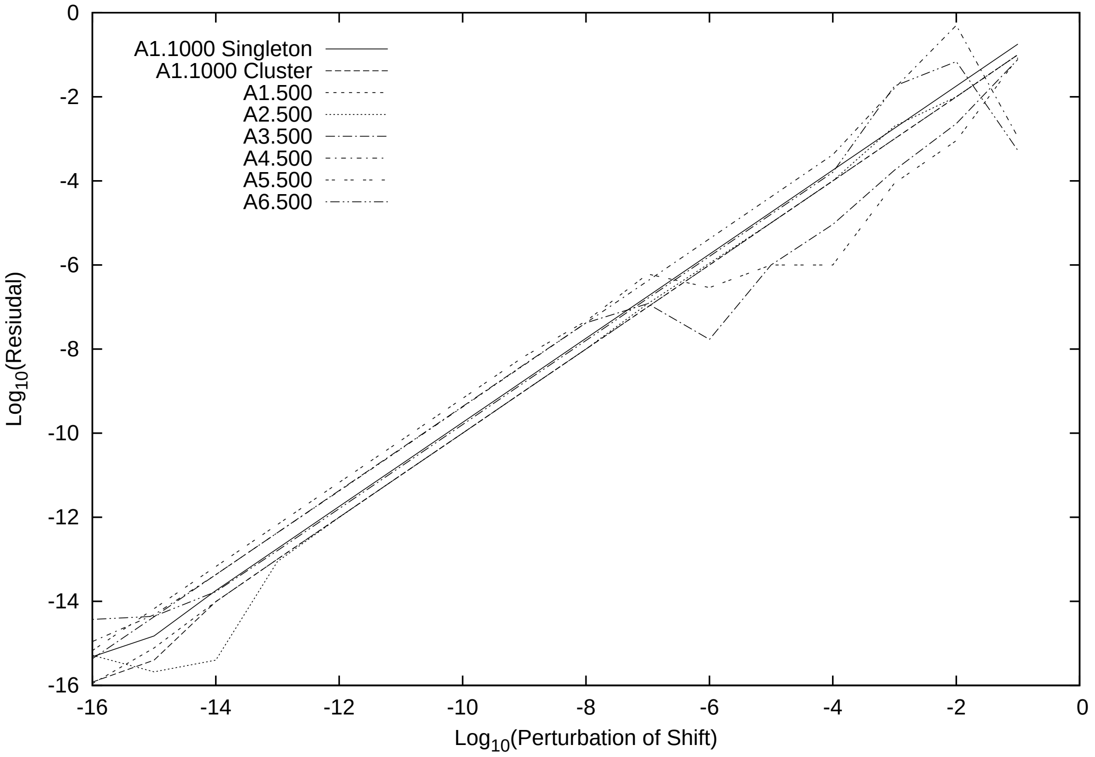

Figure 4.5: Dependence of the residual on the quality of the shift for multiple matrix types.

To demonstrate that the linear relationship between perturbation and resulting residual is also given in other matrices, a similar procedure as mentioned above was conducted for other matrix types. The resulting data is shown in Figure 4.5. However, this time only a single eigenvalue of each matrix type was selected (with the exception of matrix type A1, where one isolated eigenvalue and one eigenvalue located in a cluster of eigenvalues were selected as shifts) and the perturbations were only conducted in negative direction (i.e., the amount of the perturbation of the shift was always subtracted from the true eigenvalue). Although the data lines are more noisy than in the previous plot (due to limited sampling) a linear relationship between perturbation of the shift and resulting residual is clearly present in all matrix types. Also, there does not seem to be any relation to the presence or absence of clusters of eigenvalues in vicinity to the shift (as demonstrated by the similar results for an isolated eigenvalue in matrix type A1, denoted as "A1.1000

Singleton“ and an eigenvalue from a cluster of eigenvalues, denoted as ”A1.1000 Cluster” in Figure 4.5), as far as the residual is concerned.

In Table 4.2, we give a short overview of some noteworthy examples of eigenvector computations. The first column denotes the matrix type, the dimension n and the block size b of the matrix. The second column indicates the index of the eigenvalue that has been used as shift (all eigenvalues being indexed in increasing order). The third column gives the residual of the eigenpair obtained for the predicted starting position of the back substitution. The following column shows the ratio of the residual of the predicted starting position to the *minimal* residual obtained over all potentially possible starting positions and all factorizations (i.e., a “1.0” in this column means that strategy S_s leads to the optimal starting vector, while higher values show that there are -in theory- better starting vectors). The penultimate column gives the number of starting positions (out of 500 possible positions for the starting vector), which yield a smaller residual than the predicted starting position. Finally, the last column shows the so-called “Computational Multiplicity”, which is the number of *different* eigenvectors obtained by all possible starting positions over all factorizations for a given shift. (Note that *computational* multiplicity of different eigenvectors due the different starting positions e_s given a single shift is not equivalent with the notion of *geometric* multiplicity in eigenvalues, i.e. the dimension of the eigenspace associated to a single eigenvalue, or the number of linearly independent eigenvectors with that eigenvalue.) For this purpose, the result of each starting position with a residual below $8.5 \cdot 10^{-13}$ was compared with all eigenvectors calculated by `LAPACK/dsyevd`. If the scalar product between the result of the starting position and the Lapack-eigenvector was above 0.7 (corresponding to an angle below $\sim$ 45 degrees)[1] , the result was considered to be corresponding to this Lapack-eigenvector. Thus, computational multiplicity gives the number of different Lapack-eigenvectors encountered over all starting positions.

The first two test cases in Table 4.2 were taken from matrix type A1, which

[1] Contrary to the first experiments in this section, we do not want to evaluate the accurateness of the resulting eigenvectors using a single step of inverse iteration. Instead, this experiment is supposed to determine how many different eigenvector can *potentially* be computed with a single shift (e.g., using more steps of inverse iteration). For this purpose, the threshold for the scalar product with the eigenvectors resulting from `LAPACK/dsyevd` had to be lowered in this context.

Table 4.2: Comparison of eigenvectors computed for different matrices and different eigenvalues.

Matrix	# EV	Residual	Res./min.res.	# Better start. pos.	Mult.
A1.500-5	1	$1.0 \cdot 10^{-16}$	1.0	0	1
A1.500-5	301	$2.2 \cdot 10^{-26}$	1.0	58	65
A3.500-5	1	$2.9 \cdot 10^{-16}$	1.5	4	1
A3.500-5	301	$1.4 \cdot 10^{-16}$	1.0	1	28
A6.500-5	1	$1.1 \cdot 10^{-15}$	2.8	5	1
A6.500-5	301	$1.0 \cdot 10^{-15}$	3.1	25	1

contains an isolated eigenvalue at -1 and two clusters of eigenvalues around $\pm\varepsilon_{mach}$. While the first line shows the aforementioned isolated eigenvalue, which results in a good residual and no computational multiplicity, the second eigenvalue (301) was taken from the middle of a cluster. The computational multiplicity determined as described above is 65, which, however, does not reflect the actual size of the cluster (ca. 250 eigenvalues, since there a two clusters in A1: One at $-\varepsilon_{mach}$ and one at $+\varepsilon_{mach}$). On the other hand, this experiment demonstrates that it is (in principle) possible to obtain multiple eigenvectors from a single shift, if the gaps of the eigenvalues are relatively small. Interestingly, the residual of the eigenpair from a cluster of eigenvalues is lower than in the isolated case, which indicates that the residual is not necessarily a good measure for the quality of the result (as far as other important issues like, e.g. orthogonality of the eigenvectors) are concerned. Test cases three and four in Table 4.2 were taken from the matrix type A3, which also contains an isolated eigenvalue at -1 and geometrically distributed eigenvalues between ± 1 and $\pm\varepsilon_{mach}$. Again, using a shift corresponding to an isolated eigenvalue, as shown in line three of Table 4.2, leads to only a single eigenvector over all possible starting positions with acceptable residual (i.e., without any computational multiplicity), whereas the eigenvalue # 301 taken from a cluster of eigenvalues has a higher computational multiplicity (in the sense that different starting positions of the same shift lead to different eigenvectors). However, the computational multiplicity of the eigenvalue in row number four (A3.500-5 #301) of Table 4.2 is lower than in row number two (A1.500-5 #301), which could be explained by the lower density of eigenvalues in the cluster of matrix type A3 (because of their geometric distribution). Also, the residuals of the isolated eigenvalue and the eigenvalue from a cluster region do not differ as strongly as in the first two test cases. The final two test cases were taken from matrix type A6, whose eigenvalues are randomly distributed between ± 1. In these two cases, computational multiplicity does not pose any problem, which corresponds to our expectations (since no clusters of eigenvalues are present in A6). Interestingly, although matrix type A6 is totally unproblematic as far as computational multiplicity is concerned, the residuals of the eigenvectors are slightly higher than in the previous examples.

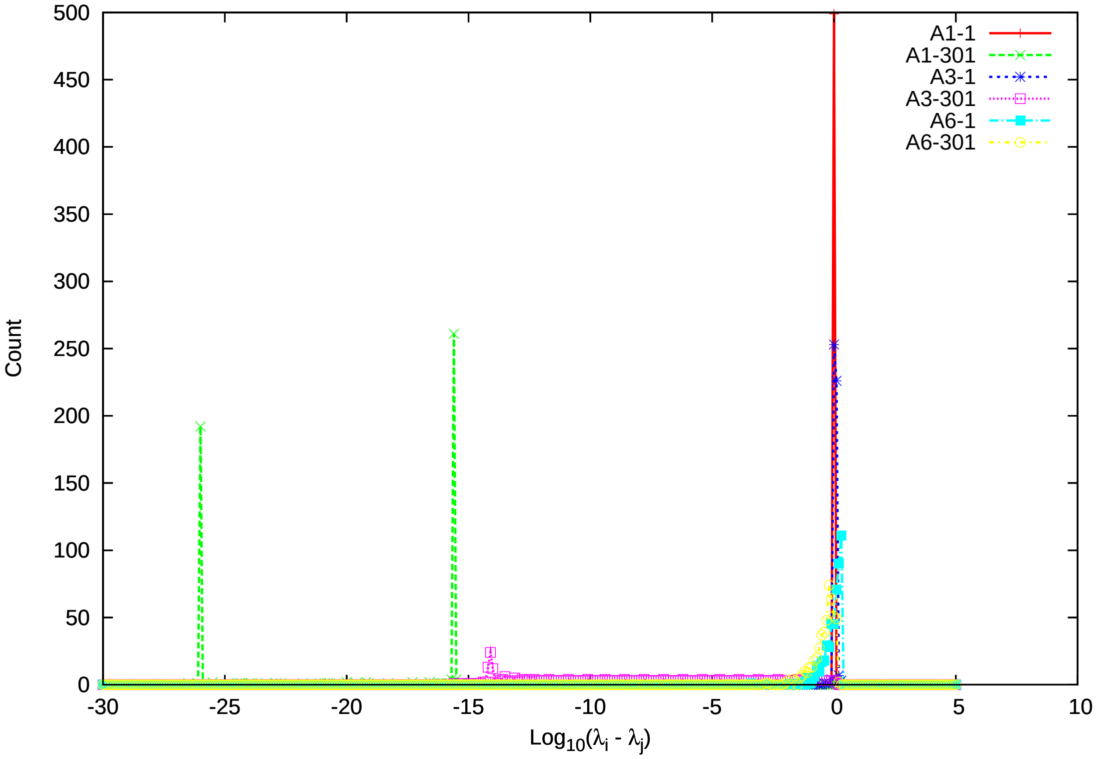

Figure 4.6: Distribution of the distances of the selected eigenvalues from the test matrices in Table 4.2 to all other eigenvalues of these test matrices (the bin width of the common logarithms is 0.1).

To better illustrate the dependency of computational multiplicity on the selection of a shift which lies within a cluster, the distances between a selected eigenvalue and all other eigenvalues in the matrix are shown in Figure 4.6. The curves are named after the matrix type and the index of the selected reference eigenvalue (e. g., the distance histogram of the isolated eigenvalue number 1 of matrix A1 is denoted as A1-1). In this figure the common logarithm of the absolute gap between the eigenvalues is plotted versus the frequency at which a certain gap occurs (for a bin width of the common logarithms of 0.1). While eigenvalues with a computational multiplicity of one (A1-1, A3-1, A6-1 and A6-301) correlate with gaps between $1.0 \cdot 10^{-1}$ and $1.0 \cdot 10^{0}$, the gaps of eigenvalues with higher computational multiplicity (A1-301) can be related to two main clusters (at $1.0 \cdot 10^{-26}$ and $1.0 \cdot 10^{-15}$) and some eigenvalues lying in between. However, in the case of A1-301, not always the eigenvectors corresponding to the closest eigenvalues (with indices around 301) are included in the “subspace“ for the shift (i.e., all different eigenvectors obtained from all possible starting positions using a single shift), but rather eigenvectors corresponding to eigenvalues with indices between 69 and 493 (with corresponding absolute gaps of $2.2 \cdot 10^{-16}$ and $3.1 \cdot 10^{-17}$). Notably, the LAPACK-eigenvector corresponding to eigenvalue 301 is not present in the subspace corresponding to the shift. This indicates that below a certain gap there are some "dominant“ eigenvectors which are easy to compute, while other eigenvectors are more difficult to obtain, even if the shift represents a good approximation to the eigenvalue. This means that below a certain gap in a cluster of eigenvalues it is not guaranteed to obtain an eigenvector for every eigenvalue in this cluster by means of inverse iteration with a general starting vector e_s (which contains only a single non-zero entry at position s). In such a case, the approach discussed in this thesis will most likely fail.

The computational multiplicity of A3-301 correlates with the size of the cluster at $\sim 1.0 \cdot 10^{-14}$, but the dimension of the subspace determined computationally is actually lower than the number of eigenvalues in this cluster and the subspace contains eigenvectors corresponding to eigenvalues between indices 208 and 323. However, in this case, not only eigenvector 301, but also the neighboring eigenvectors 300 and 302 are included in the computational multiplicity. This also indicates that small gaps lead to higher computational multiplicities, but not all eigenvectors corresponding to the eigenvalues of the cluster are necessarily represented in the computational multiplicity(i.e., all different eigenvectors obtained from all possible starting positions using a single shift).

4.6 Orthogonality of the Computed Eigenvectors

Figure 4.7 shows the common logarithms of the scalar products of the eigenvectors computed with strategy S_s for all different eigenvalues in all matrix types used (the eigenvalues are sorted in ascending order, i.e. the first eigenvalue is the smallest). The x and y axes show the corresponding indices of the eigenvectors involved, and the colour scheme is scaled logarithmically.

The larger scalar products close to the diagonals from bottom left to top right in the cases A1 to A5 indicate that closer eigenvalues tend to yield larger scalar products of their associated eigenvectors. This shows that smaller gaps lead to larger scalar products of the resulting eigenvectors, i.e., the eigenvectors are *not* orthogonal. The same holds true for eigenvalues located within a cluster (corresponding to rectangles in the plot), which is clearly illustrated in matrix types A1 and A2, where the two clusters at $\pm\varepsilon_{mach}$ can be distinguished by two rectangular patterns (associated with high scalar products, i.e., lack of orthogonality of the resulting eigenvectors). Interestingly, also the eigenvectors corresponding to eigenvalues with similar *absolute* value can have larger scalar products, as indicated by the top left to bottom right diagonals in the x-like structures for matrices A3, A4 and A5. However, whether this can be attributed to the method itself or to the way the matrix was constructed remains elusive. Matrix A6, on the other hand, shows unproblematic behavior with most scalar products in the same range for all possible combinations of eigenvectors computed. This is obviously due to the random uniform distribution (and, therefore, large relative gaps) of the eigenvalues of matrix A6.

For a visual comparison, we show the common logarithms of the associated relative gaps between pairs of eigenvalues in Figure 4.8.

By comparing Figure 4.7 and Figure 4.8 it becomes clearly discernable that the indices of eigenvalues with small relative gaps (the darkish areas in Figure 4.8) correspond to indices with large scalar products of the resulting eigenvectors (the dark areas in Figure 4.7).

Another perspective is shown in Figure 4.9, where the *mean* scalar product of each eigenvector with *all other* eigenvector results in the matrix are given. In this plot, it is easy to see that all eigenvectors in A1 and A2 (with the notable exception of the eigenvectors corresponding to the isolated eigenvalues) are not orthogonal, but

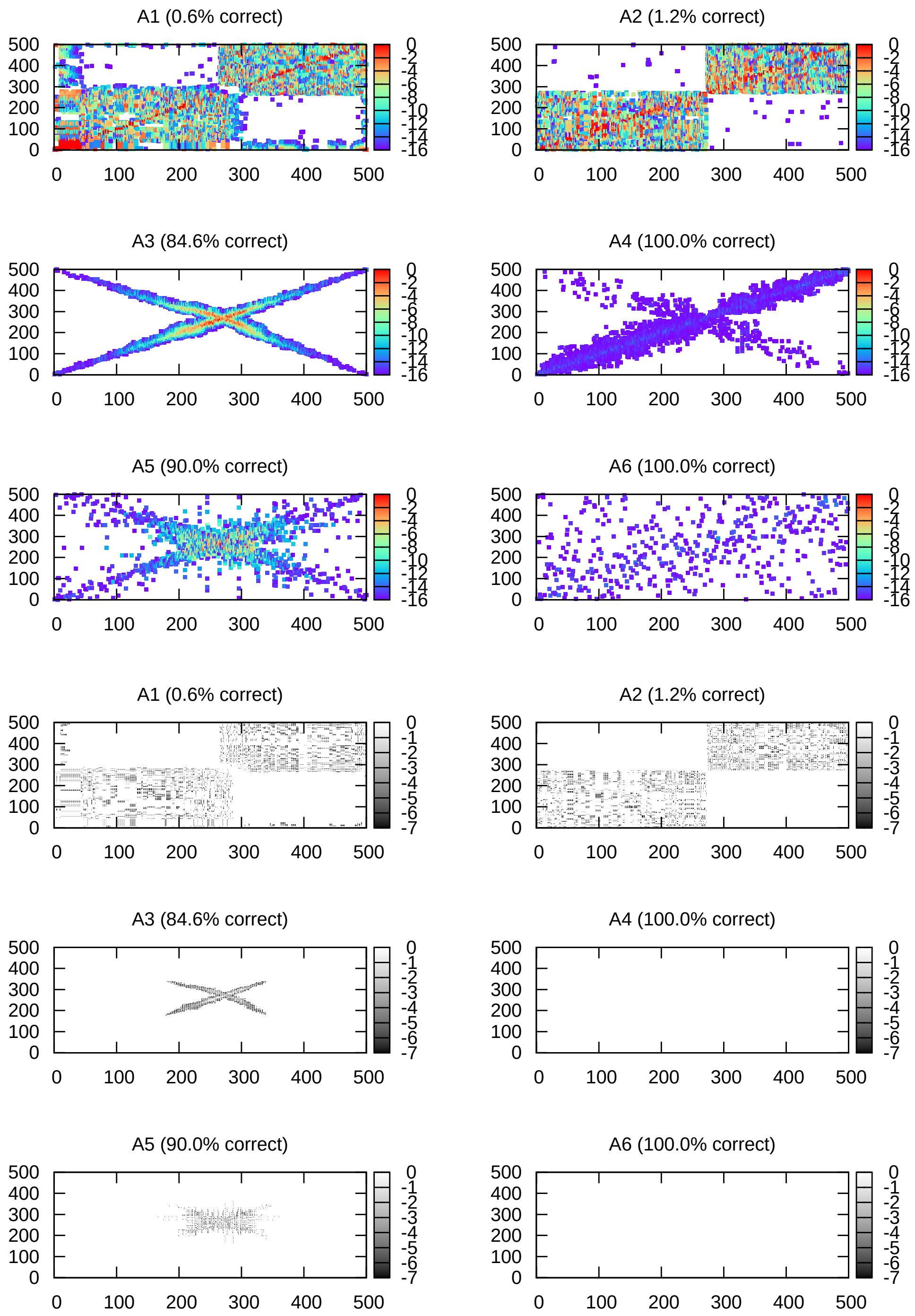

Figure 4.7: Scalar products of the computed eigenvectors for the different matrix types using two different representations. While in the upper half all scalar products above machine precision (ε_{mach}) are shown (in colour), only the most severe cases (above $\sqrt{\varepsilon_{mach}}$) are shown in the lower plots (in greyscale). Also, the percentage of correctly predicted eigenvectors is given. (An eigenvector was considered to be correct, if the scalar product with an eigenvector as computed by `LAPACK/dsyevd` was greater than 0.99)

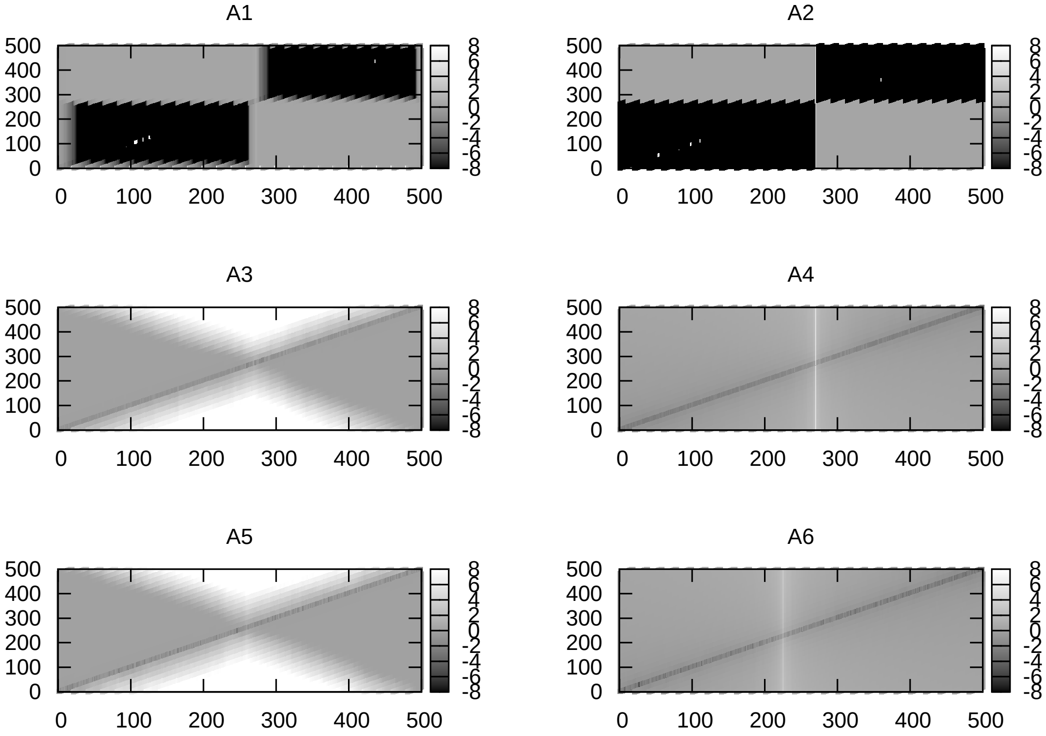

Figure 4.8: Log$_{10}$ of the relative gaps between eigenvalues for the different matrix types.

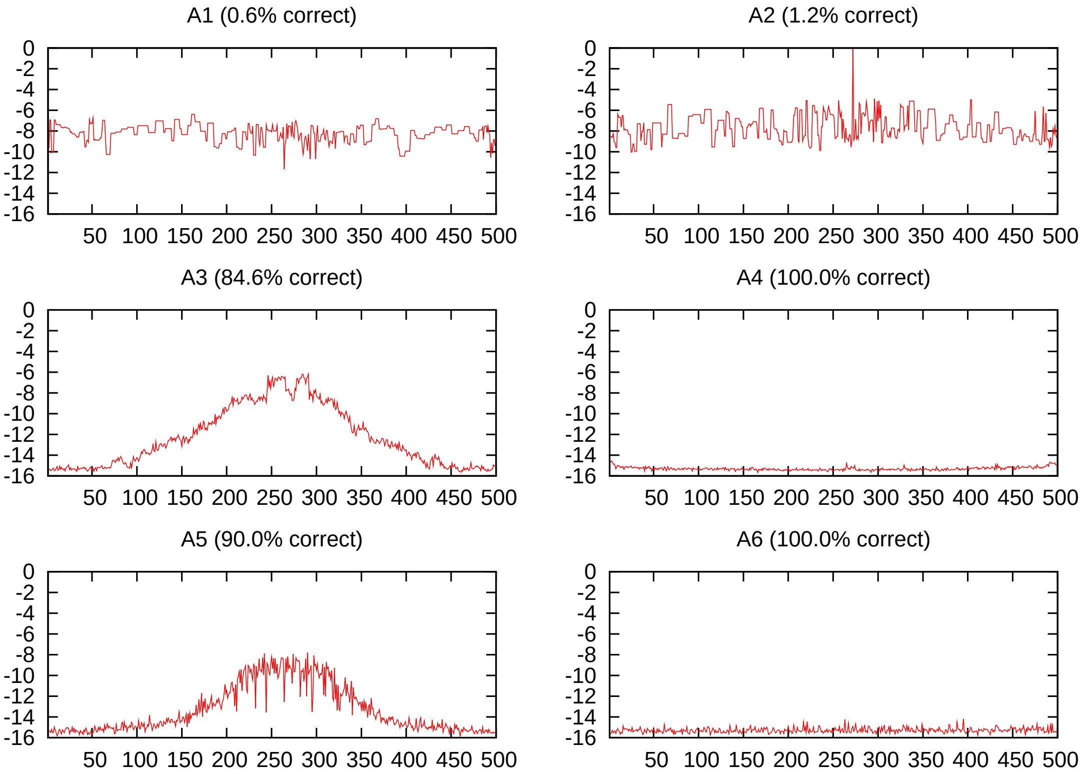

Figure 4.9: Means of the common logarithms of the scalar products of each eigenvector with all other eigenvectors for each index of the eigenvalues.

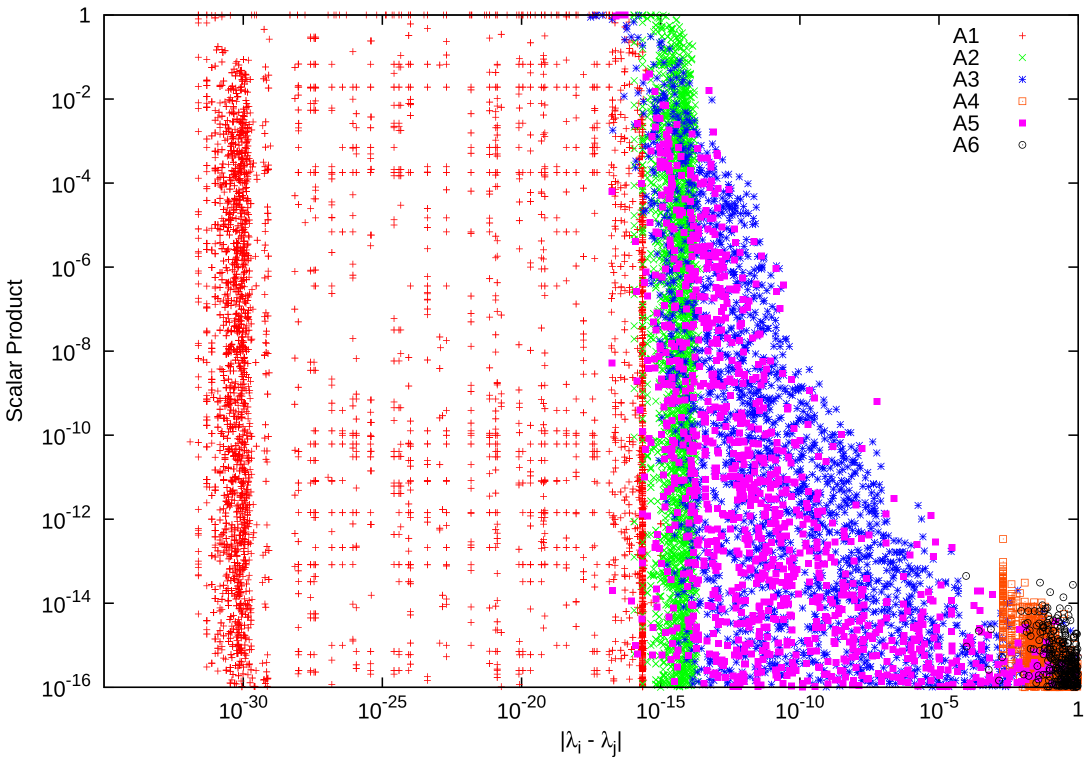

Figure 4.10: Orthogonality of computed eigenvectors depending on absolute gaps between corresponding eigenvalues.

on average show scalar products near $\sqrt{\varepsilon_{mach}}$. Also the eigenvectors corresponding to eigenvalues in the clusters of A3 and A5 are problematic, while the eigenvectors of A4 and A6 are orthogonal to almost machine precision.

Finally, Figure 4.10 plots the scalar products of the eigenvectors depending on the absolute gap between the corresponding eigenvalues. The data was collected over all matrix types and all eigenvalues for each matrix using a dimension $n = 500$. The relationship between the gap of the eigenvalues ($|\lambda_i - \lambda_j|$) and the scalar product of the corresponding eigenvectors is clearly visible. Most large scalar products are observed for small or very small absolute gaps between eigenvalues (seen on the left side of Figure 4.10), while small scalar products (orthogonal eigenvectors) are encountered when the gaps are relatively large. For the gaps between 10^{-15} and 10^{-5}, there also seems to be a linear relationship between the magnitude of the gap and the *maximum* scalar product in the plot. This gives us the possibility to predict

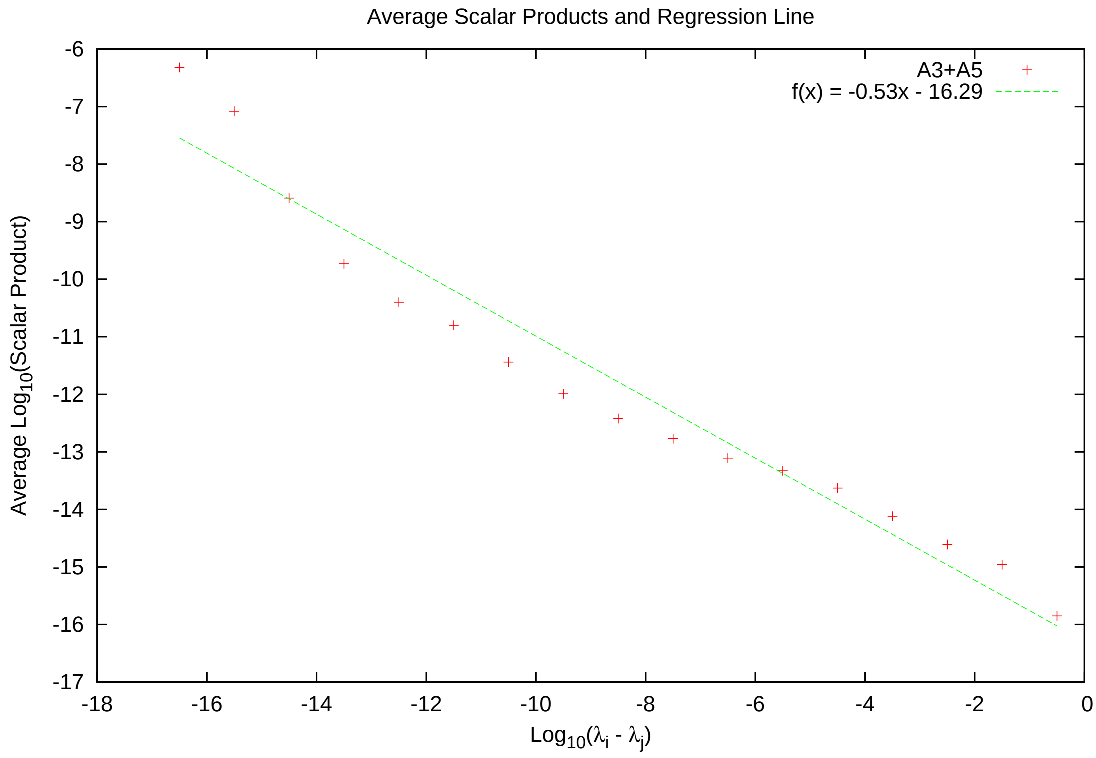

Figure 4.11: Mean scalar products given a certain gap. The regression line is given by $f(x) = -0.53x - 16.29$. $R^2 = 0.95$

the worst to be expected scalar product of two eigenvectors given the gap of the two corresponding eigenvalues.

To predict the average scalar product given a certain gap, we used data from matrix types A3 and A5 and divided the common logarithms of the gaps into bins of width 1 to calculate the average scalar product for each bin. The result is given in Figure 4.11. The data was fitted with a linear regression curve of the form

$$f(x) = -0.53x - 16.29 \tag{4.1}$$

yielding a good agreement with an R^2 of 0.95. This function can now be used to estimate the resulting scalar product based on the gap between the eigenvalues, which allows us to set a threshold for the maximum allowable scalar product (and, therefore, defining the quality of the resulting eigenvector. For example, for the average scalar product to lie below $1.0 \cdot 10^{-12}$, a minimum gap of approximately $1.0 \cdot 10^{-8}$ is required).

4.7 Subspaces Identified

As illustrated in Section 4.6, in some cases (i.e., tight clusters of eigenvectors) the eigenvector solutions obtained based on block twisted factorizations experience a loss of orthogonality. However, a set $V_T \in \mathbb{R}^{n\times m}$ of m computed eigenvectors (with $m \leq n$) obtained with block twisted factorizations could still span the same subspace as the set $V \in \mathbb{R}^{n\times m}$ of the m corresponding actual eigenvectors (e. g., if V_T is rotated). If that is the case, V can be represented as a linear combination of the columns of V_T:

$$\exists\, C \in \mathbb{R}^{m\times m}: \quad V_T C = V \tag{4.2}$$

In order to test this hypothesis, we conduct a QR factorization of V_T

$$V_T = Q \begin{pmatrix} R \\ 0 \end{pmatrix}, \tag{4.3}$$

yielding orthogonal $Q \in \mathbb{R}^{n\times n}$ and upper triangular $R \in \mathbb{R}^{m\times m}$. By inserting (4.3) into (4.2), we obtain

$$Q \begin{pmatrix} R \\ 0 \end{pmatrix} C = V.$$

As $QQ^T = I$, we can reformulate this into

$$\begin{pmatrix} RC \\ 0 \end{pmatrix} = Q^T V. \tag{4.4}$$

Consequently denoting $U = Q^T V$, the norm of the submatrix $U(m+1:n, 1:m)$ (which is supposed to equal zero) is a measure for the "incompleteness" of the subspace spanned by V_T relative to the one spanned by V.

In Figure 4.12, $\|U(m+1:n, 1:m)\|_1$ for different sets of eigenvectors in all matrix types are shown. For the data shown in this figure, each set V_T contains five eigenvectors (m = b = 5) corresponding to the respective neighboring eigenvalues in ascending order (e.g., position one in Figure 4.12 denotes a set of the first five eigenvectors of the matrix, while position 11 denotes a set of eigenvectors 11 to 15). The corresponding set V was calculated with `LAPACK/dsyevd`. Along the x axis, the indices of the first eigenvector in each set are shown. If V_T did not have full rank (using the `MATLAB` definition of rank), the same eigenvector was computed for different eigenvalues. Therefore, the results for $\|U(m+1:n, 1:m)\|_1$ are not

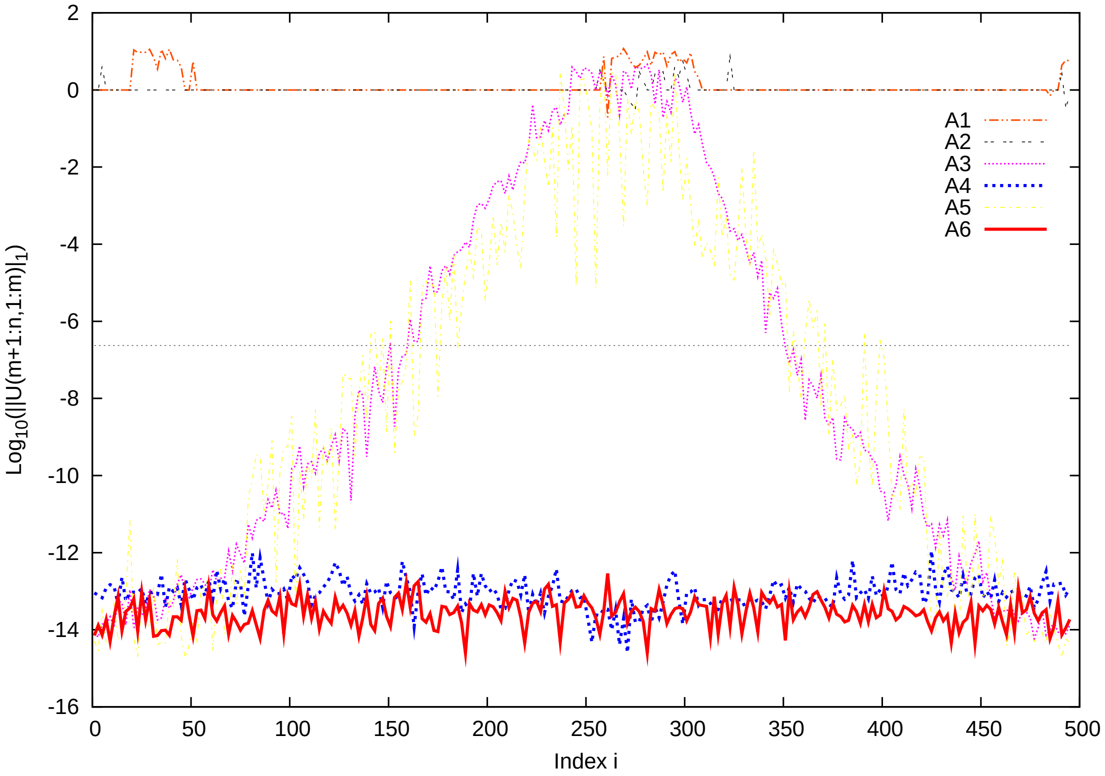

Figure 4.12: Overview of $\|U(m+1 : n, 1 : m)\|_1$ for different sets of five eigenvectors in all matrix types

meaningful. In such cases $\|U(m+1:n,1:m)\|_1$ was (arbitrarily) set to 1. The dotted line in the middle of the plot shows $\sqrt{n\varepsilon_{mach}}$ as a reference line.

For matrix types A1 and A2, V_T most of the time was not of full rank (thus forming a flat line at zero in Figure 4.12), indicating that several shifts in tight clusters of eigenvalues lead to the same eigenvector. The opposite can be seen in matrix types A4 and A6, where the eigenvalues are not clustered, as reflected by the small values (around $n * \varepsilon_{mach}$, with $n = 500$) of $\|U(m+1:n,1:m)\|_1$. In A3 and A5, $\|U(m+1:n,1:m)\|_1$ increases with decreasing size of the gaps between the eigenvalues. This further underlines the results of the previous sections, as it shows directly that the resulting eigenvectors for eigenvalues in clusters of eigenvalues are not orthogonal and, in cases of very tight clusters, the same eigenvector is obtained for multiple eigenvalues.

4.8 Use of Blocked Strategies for the Determination of Eigenvectors Corresponding to Multiple Eigenvalues

One potential advantage of blocked strategies (e.g., S_{b4}) would be a possible treatment of the problem of multiple eigenvalues in banded matrices. In banded matrices multiplicities of eigenvalues up to the block size b can occur (i.e., one eigenvalue can have up to b associated eigenvectors) . By using scalar methods (which define a single starting position for the starting vector in inverse iteration), we can only compute one vector from the multidimensional subspace of this multiple eigenvalue. However, by using a blocked approach, we have b possible starting vectors at our disposal, once we identified a suitable block.

To check this possibility, a banded matrix with $n = 500$ and a half-bandwith of five (which can be represented by a block tridiagonal matrix with block size $b = 5$) was created which, in double precision, yields an eigenvalue of multiplicity five. Such a matrix can be produced by constructing a diagonal matrix with the desired eigenvalues (in this case equally spaced eigenvalues between -1 and $+1$, whereof the last b eigenvalues were set to $+1$), which is then multiplied with an orthogonal matrix. This results in a full matrix, which can be reduced to a banded matrix by using `DSYRDB`[17].

After calculating the twisted factorization leading to the block L_kU_k with minimal absolute singular value, all positions within this block were used as starting vectors for inverse iteration using the multiple eigenvalue for shift. Surprisingly, all positions lead to the same eigenvector (with residuals of $1.01 \cdot 10^{15}$, $8.36 \cdot 10^{16}$, $3.03 \cdot 10^{15}$, $8.19 \cdot 10^{16}$ and $1.15 \cdot 10^{15}$ and scalar products of 0.999260691520366, 0.999260691519269, 0.999260691519125, 0.999260691520492 and 0.999260691520351 with the corresponding eigenvector number 499 as calculated with `LAPACK/dsyevd`). From this finding we infer that blocked strategies are also not able to cope with multiple eigenvalues.

4.9 Runtimes

Finally, we evaluate the execution times of the eigenvector computation based on block twisted factorizations. For this purpose, the eigenvectors of three different eigenvalues in a matrix were calculated with four different processes:

1. **Method BTW** based on block twisted factorizations as computed with the routine DSYBTEV (see Chapter 5): Given an eigenvalue of $W(p)$, this method computes all twisted factorizations of $W(p) - \hat{\lambda}I$ and then selects—according to the strategy S_s introduced in Section 3—the twisted factorization with the smallest diagonal element for performing back substitution (i. e., one step of inverse iteration) in order to compute an eigenvector corresponding to $\hat{\lambda}$.

2. **Reference method M1** - standard tridiagonalization followed by inverse iteration: Tridiagonalize the matrix with `LAPACK/dsytrd` and then calculate all n eigenvectors with `LAPACK/dstein`, which calculates the eigenvectors only. By dividing the total runtime by the dimension n, we obtain a virtual mean time spent for the calculation of a single eigenvector.

3. **Reference method M2** - standard tridiagonalization followed by divide-and-conquer: The routine `LAPACK/dsyevd` first tridiagonalizes the matrix using `LAPACK/dsytrd`, and then uses tridiagonal divide-and-conquer to compute

eigenvalues and eigenvectors of the tridiagonal matrix. By subtracting the time spent for the calculation of the eigenvalues and then dividing the total runtime by the dimension n, we obtain a virtual mean time spent for the calculation of a single eigenvector.

4. **Reference method M3** - standard tridiagonalization followed by relatively robust representations: Tridiagonalize the matrix with `LAPACK/dsytrd` and then compute both eigenvalues and eigenvectors using `LAPACK/dstegr`. By dividing the total runtime by the dimension n, we again obtain a virtual mean time spent for the calculation of a single eigenpair (Note that in this process the time spent for computing the eigenvalues is almost negligible and amounts to approximately 1% of the time spent in `LAPACK/dstegr`).

5. **Reference method M4** - band reduction to tridiagonal form followed by relatively robust representations: Tridiagonalize the matrix with `SBR/dsytrd` from the SBR toolbox for successive band reduction [17] and compute both eigenvalues and eigenvectors using `LAPACK/dstegr`. By dividing the total runtime by the dimension n, we again obtain a virtual mean time spent for the calculation of a single eigenvector (This approach differs from method M3 by the tridiagonalization process employed). .

Figures 4.13 and 4.14 show the *average* time spent for the computation of a single eigenvector, if all eigenvectors have to be computed. However, it shall be noted that all reference methods (except for the new BTW method) rely on a prior tridiagonalization of the matrix. While the computation of eigenvectors of tridiagonal matrices can be conducted very swiftly, the prior tridiagonalization step constitutes the most expensive part of the whole process. Since the tridiagonalization can neither be omitted nor satisfactorily divided or parallelized, the computation of a single eigenvector is thus almost as expensive as the calculation of all eigenvectors (i.e., the plotted times for the calculation of a single eigenvector in all reference methods are *virtual*). If only a particular eigenvector is desired, the discussed BTW method is in all examples several orders of magnitude faster, since it does not require a prior tridiagonalization.

Test Data and Hardware Used

Since the computational cost of the five processes compared does not depend on the matrix type, we used random block tridiagonal matrices with varying dimensions n and varying block sizes b. The experiments were performed using an Intel Pentium 4 with 3.00GHz and 1 GB of memory.

Runtimes for Varying Block Sizes

In Figure 4.13, the relation between block size b and mean execution time of three runs with different shifts for the four different methods is shown for matrices with a fixed dimension $n = 6000$.

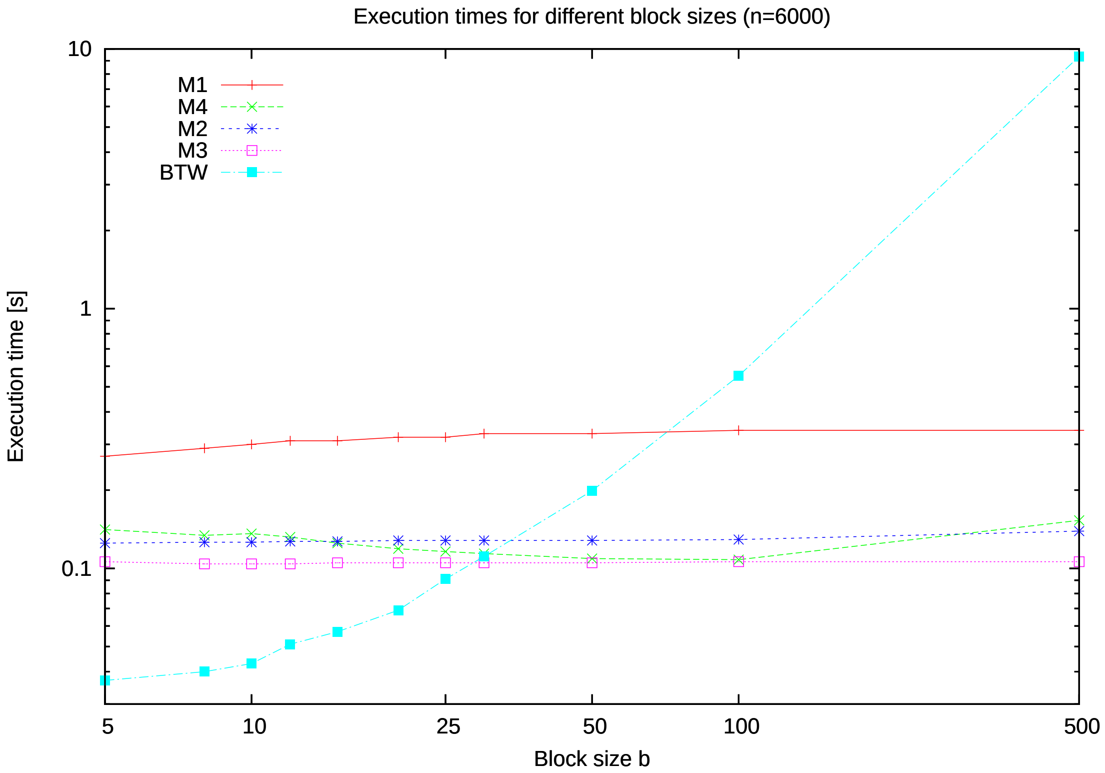

Figure 4.13: Mean execution times in seconds for varying block sizes using a random square block tridiagonal matrix of dimension $n = 6000$

The BTW(S_s) method is obviously the fastest up to block sizes around 30. However, with increasing block size, the computational cost of the current implementation of the computation of the twisted factorizations increases rapidly, while the execution times of all other methods basically does not change with the block size (except for small variations of the tridiagonalization process with the routine `SBR/dsbrdt`). Further improvements in the computation of the twisted factorizations are expected to extend the range of block sizes where BTW(S_s) provides a competitive alternative to existing methods.

Runtimes for Varying Problem Sizes

Figure 4.14 illustrates mean execution times in seconds for varying dimensions n for different test matrices with a fixed block size $b = 10$.

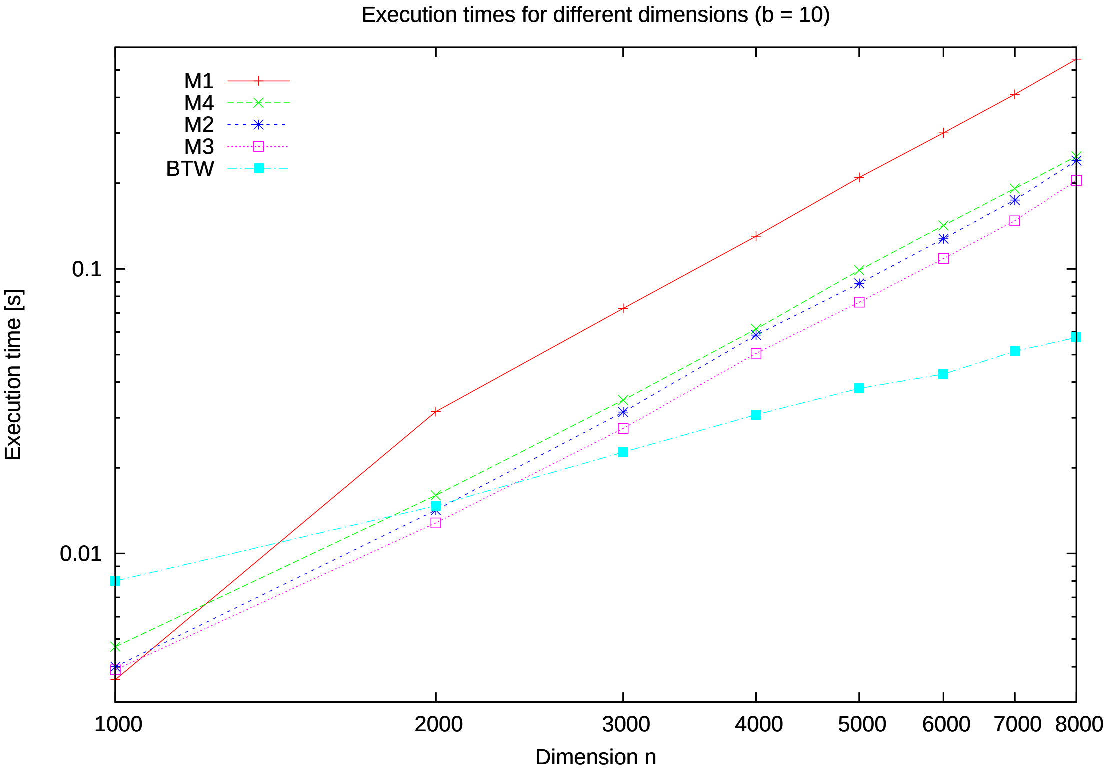

Figure 4.14: Mean execution times in seconds for varying problem sizes using a random square block tridiagonal matrix with block size $b = 10$

While for small matrixes the computational costs of the algorithm described are slightly higher, it is obvious that asymptotically BTW becomes more efficient than the other methods, in larger matrices surpassing the computational efficiency of the other methods many times over. (An advantage, which is especially important in the field of Scientific Computing, where computations involving enormous matrices are more the rule than the exception). For the current (not specifically optimized) implementation, the break-even point for $b = 10$ is between $n = 2000$ and $n = 3000$. The execution time of the current implementation of BTW(S_s) is clearly dominated by the computation of the twisted factorizations of shifted $W(p)$, which might become further optimized by clever incorporations of the shift or by modifying the code to exploit certain matrix structures, while, on the other hand, the scope for improvements in (the computationally less demanding) back substitution is very narrow.

Chapter 5

Implementation

The goal of this chapter is to give an overview of the code. In Section 5.1, we describe the highest level implementation of the code, characterize the data structures to be used and also give an outline of the sequence of the computational and auxiliary routines, which are described in more detail in Sections 5.2, 5.3, 5.4 and 5.5. Also, since the code heavily relies on the performance of BLAS[18] and LAPACK[19], the routines employed are described in Section 5.6.

The algorithm computes only the eigenvectors of quadratic block tridiagonal matrices (thus the eigenvalues have to be obtained by other means), however, the use of both symmetric and unsymmetric matrices is (in principle) admissible. By itself, it is an improvement of the standard inverse iteration which employs twisted factorizations for the determination of a good starting vector. (The connections between the twisted factorization and the inverse of the matrix are discussed elsewhere[14, 10])

To obtain reliable results for the eigenvector, two conditions must be met:

- The eigenvalues must be determined to high accuracy
- The gap between adjacent eigenvalues must be large enough (as specified by Equation 4.1)

5.1 The main program BTEV

The highest level implementation is the Fortran90 program "BTEV". All subroutines necessary are collected in the module "STUFF", which is collocated in the file btev.f90. The main objectives of BTEV are as follows:

1. Preparation of the data structures as shown in Table 5.1.

2. Reading a previously saved block tridiagonal matrix from a file, or generating a block tridiagonal matrix filled with random numbers between 0 and 1.

3. Calculating the eigenvalues with `LAPACK/dsyevd`

4. Calling the driver routine DSYBTEV for the computation of a specified eigenvector

5. Comparison and analysis of the results

When executed, BTEV expects two command line arguments, which offer two possibilities for the input:

- Read the matrix from a file. In that case, the first command line argument should be "-f", while the second command line argument should specify the file name. (For further information on the matrix data format, see the following Subsection 5.1.1)

- Generate a matrix filled with random numbers. In this case, the first command line argument is the number of main diagonal blocks (p), while the second command line argument states the number of columns/rows of per quadratic block (b). The dimension n of the matrix is given by $n = p \cdot b$.

In addition to the two command line arguments, an integer number has to be passed on from STDIN, which specifies the index of the eigenvector to be calculated by BTEV (the eigenvalues being sorted in ascending order).

5.1.1 Matrix data format

BTEV offers the ability to process previously saved matrices in a certain block-based format. This file format is one-dimensional.

- The first entry (which is integer) specifies the number of main diagonal blocks p.

- The following p integer entries specify the block size b of each block separately (however, until now, only equal blocksizes are supported by BTEV). These variables define the general shape of the block tridiagonal matrix.

Table 5.1: Main data structures

Variable name	Type	Shape	Function
COUNT	Int	1×1	Number of blocks in the main diagonal (p)
STEPS	Int	1×1	Number of columns/rows per block (b)
CS	Int	1×1	Matrix dimension (n = p * b)
pivot	Int	$n \times p \times 3$	Saves the pivoting vectors of all LU factorizations. The first dimension saves the pivot indices as returned by `LAPACK/dgetrf`, the second dimension denotes the number of the block, while the last dimension states whether the block belongs to a forward/backward or twisted factorization
A	Double	$b \times b \times (p - 1) \times 3$	Before DSYBTEV, contains the *subdiagonal* blocks of the original (unshifted) matrix, while after DSYBTEV (on output), contains the subdiagonal blocks (denoted as M in the introduction) of all twisted factorizations for a specified shift. The first two dimensions contain the block, the third dimension denotes the number of the block, while the last dimension states whether the block belongs to a forward/backward or twisted factorization
B	Double	$b \times b \times p \times 3$	Before DSYBTEV, contains the *main diagonal* blocks of the original (unshifted) matrix, while after DSYBTEV (on output), contains the main diagonal blocks of all twisted factorizations (L and U being combined as in `LAPACK/dgetrf`) for a specified shift. The first two dimensions contain the block, the third dimension denotes the number of the block, while the last dimension states whether the block belongs to a forward/backward or twisted factorization
C	Double	$b \times b \times (p - 1) \times 3$	Before DSYBTEV, contains the *superdiagonal* blocks of the original (unshifted) matrix, while after DSYBTEV (on output), contains the superdiagonal blocks (denoted as N in the introduction) of all twisted factorizations for a specified shift. The first two dimensions contain the block, the third dimension denotes the number of the block, while the last dimension states whether the block belongs to a forward/backward or twisted factorization
ev	Double	$n \times 1$	The computed eigenvector

- Next, the $b \times b$ entries of the first main diagonal block follow (the block being saved in column-major-order in case of asymmetric matrices). After that, the other $p - 1$ main diagonal blocks (of size $b \times b$) follow in the same (column-major-order) fashion.

- After the p *main diagonal* blocks outlined above, the $p - 1$ *subdiagonal* blocks follow.

- Finally, the $p - 1$ *superdiagonal* blocks are read.

Thus, in total, each file should contain $1 + p + (3 \times p - 2) \times b^2$ entries.

5.1.2 Workspace requirements

The main portion of the workspace is required for real variables, while the integer workspace is almost negligible (being restriced to scalars and the $3 \times n$ pivoting vector). For the calculation of the twisted factorizations, a workspace of approximately $9 \times b \times n$ is necessary, since all factorizations have to be saved for later use. In addition, the eigenvalue calculations with `LAPACK/dsyevd` temporarily require a double array of $1 + 6 \times n + 2 \times n^2$ and an integer workspace of $3 + 5 \times n$.

5.2 DSYBTEV

DSYBTEV is the main driver for the algorithm. It calculates the eigenvector for a specified eigenvalue. Thus, in addition to the data shown in Table 5.1, which includes the sub- main and superdiagonal blocks, the pivoting vector and an array for the eigenvector, DSYBTEV also needs a shift to be passed on (which is subtracted from the main diagonal elements within the routine).

The two major components of DSYBTEV are the calculation of all twisted factorizations, as implemented in the subroutine DSYBTTWF (described in Subsection 5.3) and the back substitution as implemented in DSYBTBS (See Section 5.4). Based on the results from DSYBTTWF, the driver routine DSYBTEV determines the starting position for the back substitution and also incorporates the correct pivoting before starting DSYBTBS. Finally, the eigenvector is scaled to a length of 1.

5.3 DSYBTTWF

DSYBTTWF is the major subroutine of DSYBTEV. For a block tridiagonal matrix, DSYBTTWF will calculate all twisted factorizations, as outlined in Section 2.1. In this process it employs `LAPACK/dgetrf` to factorize the main diagonal blocks and `LAPACK/dtrsm` to solve the equations for the sub- and superdiagonal blocks. The computational costs of the LU-factorizations amount to $O(n^3)$ operations (the partial pivoting adds a quadratic term only), thus constituting the main bottleneck in the whole process of computing the eigenvectors.

5.4 DSYBTBS

DSYBTBS performs the back substitution and, for optimal performance, mainly relies on `LAPACK/dtrsm` to obtain the solution to the system of equations by employing a block-wise procedure. To determine the starting position for the back substitution, is relies on the following parameters to be passed on: the index of the twisted factorization to be employed (in the program denoted as *fac*), the block with the minimal diagonal element (denoted as *blocks*. This corresponds to strategy S_s) and its exact position within the block (denoted as *ele*). Contrary to the LU-factorization, the time required for back substitution is $O(n^2)$ only.

5.5 Auxiliary routines

In addition to DSYBTTWF and DSYBTBS a number of additional routines where necessary:

- **PIVOTING**: Applies the pivoting to matrix according to the corresponding pivoting vector, as obtained from `LAPACK/dgetrf`.

- **ANTIPIVOTING**: Reverts the pivoting of matrix according to the corresponding pivoting vector from `LAPACK/dgetrf`, thus re-establishing its original form

- **ANTIPIVOTINGVEC**: A more efficient version of ANTIPIVOTING, intended for vectors only

- **UNITESUBS2TOTAL**: Constructs a full n × n matrix from the sub- main- and superdiagonal blocks (A,B,C). Intended for testing purposes and necessary for the application of `LAPACK/dsyevd`, since all other procedures employ the blocked data format.

5.6 BLAS/LAPACK routines

5.6.1 BLAS

BLAS (Basic Linear Algebra Subprograms)[20] is an efficient, portable, and widely available library of standard routines for very fundamental vector and matrix operations using various data types. BLAS was first published in 1979[18], and enjoys widespread use in high-performance supercomputing, since many producers of hardware also offer highly optimized implementations of BLAS. (This can probably be attributed to the fact that benchmarks for floating point computing power, like, e.g., LINPACK[21], which make use of BLAS, usually serve as a measure for ranking supercomputers in the TOP500 list of the world's fastest computers.) For improved performance, ATLAS (Automatically Tuned Linear Algebra Software)[22] has been employed to generate an optimized BLAS library.

The BLAS functionality can be divided into three levels:

1. Scalar, vector and vector-vector operations of the form $y \leftarrow \alpha x + y$ (like scalar dot products and vector norms)

2. Matrix-vector operations of the form $y \leftarrow \alpha Ax + \beta y$ (like solving systems of linear equations)

3. Matrix-matrix operations of the form $C \leftarrow \alpha AB + \beta C$ (like the General Matrix Multiply operation)

Two BLAS routines were regularly used in the code:

- xGEMV
  Performs a matrix-vector multiplication of the form $y = \alpha Ax + \beta y$, which, in our case, was always performed using double-precision:
  DGEMV(TRANS,M,N,ALPHA,A,LDA,X,INCX,BETA,Y,INCY)

where TRANS specifies whether A is transposed, M gives the number of rows and N the number of columns of the matrix A. LDA specifies the leading dimension of the matrix A in the memory, while INCX and INCY specify the increment for the elements of the vectors x and y.

- xGEMM

  Performs a matrix-matrix multiplication of the form $C \leftarrow \alpha AB + \beta C$. Specifically, the double-precision version has been employed:

  DGEMM (TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC)

  where TRANSA and TRANSB state whether the matrices A and B are transposed, M is the number of rows in matrix A and C. N is the number of columns in matrix B and C. K is the number of columns in matrix A and rows in matrix B. LDA, LDB and LDC specify the leading dimension of the matrices in the memory.

5.6.2 LAPACK

LAPACK (Linear Algebra PACKage)[19, 23] is a linear algebra library written in Fortran90 that contains routines for a plethora of numerical problems like solving systems of simultaneous linear equations, least square solutions of linear systems of equations, eigenpair calculation, and singular value decomposition for multiple data (real/complex, single/double precision) and matrix types (e.g., band or tridiagonal). Also, several kinds of matrix factorizations (such as LU, QR, SVD, Cholesky and Schur decomposition) are included. LAPACK exploits the functionality of BLAS, which allows substantial performance gains.

LAPACK routines follow a characteristic naming convention in the form of *pmmaaa*, where *p* denotes the data type (S, D stand for real, C and Z for complex single and double precision arithmetic). *mm* is a two-letter code describing the form of the matrix (e.g., GE for a general, unsymmetric matrix, TR for a triangular matrix and SY for a symmetric matrix). The last three letters *aaa* describe the actual algorithm (e.g., EVD for eigenvalue decomposition)

The following LAPACK routines were employed:

- DGETRF

  DGETRF computes an LU factorization of a general matrix A using partial pivoting with row interchanges (see Chapter 1). The factorization has the form $A = PLU$, where P is a permutation matrix, L a is lower triangular matrix with unit diagonal elements, and U is a upper triangular matrix. The routine is called with the command:

  DGETRF(M, N, A, LDA, IPIV, INFO)

  where M and N are the number of rows/columns of the matrix A, LDA is its associated leading dimension, IPIV is a vector containing the pivot indices and INFO returns some useful information in case of errors. The original matrix A is destroyed in the process and replaced by L and U from the factorization (Note that it is not necessary to store the diagonal elements of L explicitly, since they are *by definition* unit.)

- DTRSM

  DTRSM solves one of the matrix equations

  $AX = \alpha B$ or $XA = \alpha B$

  where α is a scalar, X and B are m × n matrices, while A is a upper or lower triangular matrix, as obtained from `LAPACK/dgetrf`. The routine is called with the command:

  DTRSM(SIDE,UPLO,TRANSA,DIAG,M,N,ALPHA,A,LDA,B,LDB)

  where SIDE specifies whether A appears on the left or right of X, UPLO specifies whether A is upper or lower triangular, TRANSA defines whether A is transposed, DIAG whether A is unit triangular, M and N are the number of rows/columns of B, while LDA and LDB denote the leading dimensions of A and B.

- DSYEVD

  DSYEVD computes all eigenvalues and, optionally, eigenvectors of a real symmetric matrix A. (This routine has been employed for the determination of the eigenvalues of $W(p)$ for the subsequent determination of the eigenvalues) In case eigenvectors are to be computed, a divide and conquer algorithm is applied. The routine is called with the command:

  DSYEVD(JOBZ, UPLO, N, A, LDA, W, WORK, LWORK, IWORK, LI-

WORK, INFO)
where JOBZ defines whether eigenvectors are desired, UPLO specifies, whether A is stored in the upper or lower triangle, N is the order of the matrix A, LDA is the leading dimension of A, W is the vector containing the eigenvalues after successful completion, WORK is an double precision array of size LWORK, IWORK is an integer array of size LIWORK. After completion, the matrix A is replaced by the eigenvectors. DSYEVD requires 1 + 6n + 2n^2 double and 3 + 5n integer space.

- DGESVD
  DGESVD computes the singular value decomposition (SVD, as necessary for strategy S_{b4}) of a real M-by-N matrix A, optionally computing the left and/or right singular vectors: $A = U\Sigma V^T$
  where Σ is an M-by-N matrix which is zero except for its min(m,n) diagonal elements, U is an M-by-M orthogonal matrix, while V is an N-by-N orthogonal matrix. The singular values in Σ are real and non-negative, and given in the diagonal in descending order.
  DGESVD(JOBU, JOBVT, M, N, A, LDA, S, U, LDU, VT, LDVT, WORK, LWORK, INFO)
  JOBU and JOBVT specify, whether all or only parts of U/V^T shall be computed, S stands for Σ, LDA, LDU, LDVT are the leading dimension of the associated matrices, while WORK is a double array of size 5∗MIN(M,N).

5.7 Compiling and Usage

All routines necessary are located in the file “btev.f90”. However, also two libraries are necessary for compilation: LAPACK and BLAS. Using, e.g., the GNU Fortran Compiler on a LINUX system, it is possible to compile btev.f90 with the following command:

```
gfortran btev.f90 -lblas -llapack -o exec
```

After successful compilation, it is possible to calculate a specified eigenvector of a matrix with the command

echo *index* | ./exec -f *matrixfile*

Where *index* gives the number of the eigenvector to be calculated (all eigenvalues are indexed in increasing order), while *matrixfile* gives the location of a file, which contains the target matrix in the format specified in Section 5.1 (e.g., “echo ”10” | ./test -f TW50.A1.1000“ will calculate the eigenvector corresponding to the 10th-smallest eigenvalue of a matrix specified in the file TW50.A1.1000)

Without the parameter ”-f“ as the first command line argument, a block tridiagonal matrix filled with random numbers between 0 and 1 will be generated:

echo *index* | ./exec *blocks width*

Here, *blocks* gives the number of main diagonal blocks (p) and *width* gives the number of columns/rows per block (b), while *index* again defines the index of the target eigenvector (e.g., echo ”1” | ./test 10 5 will generate an 500×500 matrix with ten diagonal blocks and nine super/subdiagonal blocks of width five and calculates the eigenvector corresponding to the smallest eigenvalue).

Chapter 6

Conclusions

6.1 Conclusions

In this thesis we described the basic principles of an algorithm for computing block twisted factorizations of a block tridiagonal (or band) matrix $W(p) = LU$. Furthermore, we showed the connections of the twisted factorizations to the inverse of the matrix ($W(p)^{-1}$) and a method to compute the eigenvectors of a matrix ($W(p) - \lambda I$) in a single step of inverse iteration, given a good approximation to an eigenvalue λ. This algorithm for computing eigenvectors of a block tridiagonal matrix was implemented and empirically evaluated.

We first addressed the central algorithmic question of how to choose an appropriate starting vector for the inverse iteration process. For this purpose, we motivated and compared several strategies for their effectiveness in terms of numerical accuracy and computational performance. Our data suggests that two strategies are viable: The scalar strategy S_s based on the minimal main diagonal element of U_k, and the block strategy S_{b4} based on the minimal singular value of the twisted block $L_k U_k$. This finding was further confirmed by calculating the residuals for a number of eigenvectors in various matrix types.

By considering all possible starting vectors of the form $e_s(s) = 1$, $\|e_s\| = 1$ (for $s = 1, \ldots, n$) we could also demonstrate that, in some cases, (i.e. in tight clusters of eigenvalues) not a single starting vector e_s is able to produce the correct eigenvector in a single step of inverse iteration (even though the residual of the resulting eigenvectors is very low). However, the data also suggests that for all non-pathological cases, the strategies S_s and S_{b4} return results, whose quality is comparable to the most accurate eigenvectors possible with such kinds of starting vectors and a single step of inverse iteration. To deal with the problem of wrongly returned eigenvectors, we could show that problematic cases can be predicted on basis of their gap to the neighboring eigenvalues. This allows us to turn to alternative techniques in such cases.

Finally, we tested the performance of the algorithm and compared it to more established techniques for the determination of eigenvectors. While the computational costs of the current version of our algorithm strongly depends on the block size b, we could show that the dependency on the dimension n of the matrix is more favorable than in all other methods. Thus, for larger matrices (and relatively small block sizes), the calculation of eigenvectors with twisted factorizations is several times more efficient than methods which rely on prior tridiagonalization.

6.2 Future Work

In addition to employing more steps of inverse iteration, the problem of "dominant" eigenvectors (i.e., eigenvectors, which result from multiple shifts in a cluster of eigenvalues) might be solved by using starting positions, which correspond to very small entries in the "dominant" eigenvector, but are relatively rich in the sought-after eigenvector. Such an approach would also solve the problem of multiple eigenvalues (if the dimension of the associated eigenspace to an eigenvalue is larger than one, i.e. the number of linearly independent eigenvectors with that eigenvalue is higher than one). This procedure would demand a very small modification of strategy S_s, since it would have to incorporate an additional check, whether the selected starting position is already rich in one of the eigenvectors that have already been calculated.

Another strategy could involve changing the nature of the starting vector. Based on the results for the theoretical strategy $\mathbf{S}_{\text{optevec}}$ in Section 4.3, it is clear that in some cases no starting vector of the form e_s with $e_s(s) = 1$, $\|e_s\| = 1$ is able to produce acceptable results. Instead of using e_s, it might, therefore, be fruitful to use a starting vector that contains multiple entries, whose positions correspond to a low gamma. This would mean that instead of using a single position, which is rich in the resulting eigenvector, it could be possible to use multiple rich positions. How this could be done, remains at this point unclear.

Furthermore, for improved performance on multicore structures or in the field of Supercomputing, the algorithm could be parallelized on multiple levels:

- Firstly, each shift is totally independent of each other. Thus, for a computation of all eigenvectors of a matrix (with dimension n), n different instances of DSYBTEV can be employed in parallel without disadvantageous side effects.
- Secondly, both the forward and the backward factorization are independent of each other, and, therefore, can be computed in parallel.
- Thirdly, once the forward and the backward factorization are computed, the twisted factorizations can be computed in parallel, since they only depend on the forward and the backward factorization, but not on each other. Also, once the first half of the forward and the second half of the backward factorizations are computed, some sort of pipelining is conceivable for the twisted factorizations.
- Parallelization is also possible on the level of back substitution, since in twisted factorizations the parts corresponding to the forward factorization and the parts corresponding to the backward factorization are independent.
- Finally, parallelized versions of the BLAS and LAPACK libraries (such as PBLAS and ScaLAPACK) could be employed for the parallelization of the basic linear algebra processes, which build the fundament of the algorithm (e.g., matrix multiplication, scalar LU factorization and back substitution).

We want to stress that these multi-leveled possibilities for parallelization make the block twisted factorization very powerful in comparison to other techniques for determining the eigenvectors of block tridiagonal and band matrices, which are of more scalar nature. Especially methods, which rely on prior tridiagonalizations of the matrix are very difficult to parallelize. Thus, in the light of future hardware developments (which are currently characterized by the development of higher numbers of multicores) the application of block twisted factorizations might become increasingly attractive.

Bibliography

[1] Y. Bai, W. N. Gansterer and R. C. Ward, "Block tridiagonalization of "effectively" sparse symmetric matrices", ACMTOMS, **30**, 326 (2004).

[2] C. H. Bischof, B. Lang and X. Sun, "Parallel tridiagonalization through two-step band reduction", in Proceedings of the 1994 Scalable High-Performance Computing Conference, Washington D.C., 1994, pp. 23–27.

[3] C. H. Bischof, B. Lang and X. Sun, "A framework for symmetric band reduction", ACM Trans. Math. Software, **26**, 581 (2000).

[4] W. N. Gansterer, R. C. Ward and R. P. Muller, "An extension of the divide-and-conquer method for a class of symmetric block-tridiagonal eigenproblems", ACMTOMS, **28**, 45 (2002).

[5] W. N. Gansterer, R. C. Ward, R. P. Muller and W. A. Goddard, "Computing approximate eigenpairs of symmetric block tridiagonal matrices", SISC, **25**, 65 (2003).

[6] I. S. Dhillon, B. N. Parlett and C. Voemel, "The design and implementation of the MRRR algorithm", ACM Trans. Math. Software, **32(4)**, 533 (2006).

[7] J. Demmel and W. Kahan, "Accurate singular-values of bidiagonal matrices", SIAM J. Sci. Stat. Comp., **11(5)**, 873 (1990).

[8] I. S. Dhillon and B. N. Parlett, "Multiple representations to compute orthogonal eigenvectors of symmetric tridiagonal matrices", Linear Algebra Appl., **387**, 1 (2004).

[9] K. V. Fernando, "On computing an eigenvector of a tridiagonal matrix .1. Basic results", SIAM J. Matrix Anal. A., **18(4)**, 1013 (1997).

[10] B. N. Parlett and I. S. Dhillon, “Fernando’s solution to Wilkinson’s problem: An application of double factorization”, Linear Algebra Appl., **267**, 247 (1997).

[11] B. N. Parlett, “For tridiagonals T replace T with LDL”, J. Comput. Appl. Math., **123(1-2, Sp. Iss. SI)**, 117 (2000).

[12] B. N. Parlett and I. S. Dhillon, “Relatively robust representations of symmetric tridiagonals”, Linear Algebra Appl., **309(1-3)**, 121 (2000).

[13] B. N. Parlett and O. A. Marques, “An implementation of the dqds algorithm (positive case)”, Linear Algebra Appl., **309(1-3)**, 217 (2000).

[14] C. Voemel and J. Slemons, “Twisted factorization of a banded matrix”, BIT Numerical Mathematics, **49(2)**, 433 (2009).

[15] G. H. Golub and C. F. Van Loan, Matrix Computations, Johns Hopkins University Press, Baltimore, MD, 1996, 3rd edn.

[16] I. C. F. Ipsen, “Computing an eigenvector with inverse iteration”, SIAM Rev., **39**, 254 (1997).

[17] C. H. Bischof, B. Lang and X. Sun, “Algorithm 807: The SBR toolbox—software for successive band reduction”, ACM Trans. Math. Software, **26**, 602 (2000).

[18] C. L. Lawson, R. J. Hanson, D. Kincaid and F. T. Krogh, “Basic linear algebra subprograms for FORTRAN usage”, ACM Trans. Math. Soft., **5**, 308 (1979).

[19] J. Dongarra and J. Demmel, “LAPACK - A portable high-performance numerical library for linear algebra”, Supercomputer, **8(6)**, 33 (1991).

[20] L. S. Blackford, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, M. Heroux, L. Kaufman, A. Lumsdaine, A. Petitet, R. Pozo, K. Remington and R. C. Whaley, “An updated set of Basic Linear Algebra Subprograms (BLAS)”, ACM Trans. Math. Software, **28(2)**, 135 (2002).

[21] J. J. Dongarra, “The Linpack benchmark - an explanation”, Lect. Notes Comput. Sc., **297**, 456 (1988).

[22] R. C. Whaley, A. Petitet and J. J. Dongarra, “Automated empirical optimizations of software and the ATLAS project”, Parallel Computing, **27(1-2)**, 3 (2001).

[23] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney and D. Sorensen, LAPACK Users’ Guide, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1999, 3rd edn.

Curriculum Vitae

Name: Mag. Gerhard König

Anschrift: Siegfried Esterl Gasse 15, 8160 Weiz

Geburtsdatum: 18.05.1981

Geburtsort: Weiz

Eltern:

Gerhard Alexander König, geb. 16.10.1955, Facility Manager

Margareta König (geb. Städtler), geb. 05.02.1954, Pensionistin

Ausbildung:

1987-1991 Volkschule Weiz

1991-1999 Bundesrealgymnasium Weiz

2000 Präsenzdienst

2000-2005 Studium Molekulare Biologie an der Universität Wien mit den Schwerpunkten Biochemie, Strukturbiologie und Bioinformatik. Diplomarbeit bei Prof. Stefan Boresch am Institut für Theoretische Chemie und Molekulare Strukturbiologie.

15.12.2005 Abschluss Molekulare Biologie (mit Auszeichnung)

Seit 2006 Doktoratsstudium der Naturwissenschaften (Molekulare Biologie) am Institut für Computergestützte Biologische Chemie bei Prof. Stefan Boresch und Masterstudium Scientific Computing an der Universität Wien.