Applied Research Laboratory

Technical Report

.0

19951211 048

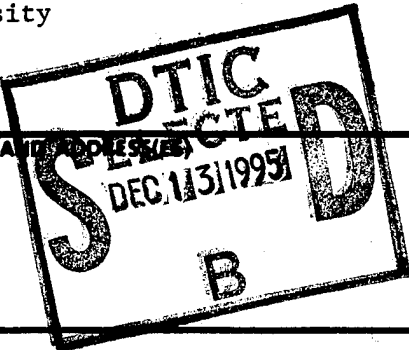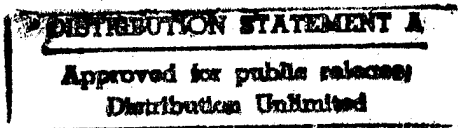# PENNSTATE

1 8 5 5

REPORT DOCUMENTATION PAGE

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	Oct. 1995	Technical Report, Dec. 93 - Oct. 95

4. TITLE AND SUBTITLE

Modifying Two-Sided Orthogonal Decompositions: Algorithms, Implementation, and Applications.

5. FUNDING NUMBERS

6. AUTHOR(S)

Peter A. Yoon

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Applied Research Laboratory
The Pennsylvania State University
P.O. Box 30
State College, PA 16804

8. PERFORMING ORGANIZATION REPORT NUMBER

TR-95-002

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Dr. Kam Ng, Code 334
Office of Naval Research
Ballston Tower 1
800 N. Quincy St.
Arlington, VA 22217-5660

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

Also submitted as the Ph.D Thesis of Peter A. Yoon to the Computer Science and Engineering Department, The Pennsylvania State University, December 1995.

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Distribution unlimited

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

In this thesis we propose several algorithms for rank-one updates and downdates to these decompositions with strong stability properties and efficient implementations on high-performance computers. We seek algorithms which only require $\mathcal{O}(n^2)$ operations per update or downdate unlike recomputing the two-sided orthogonal decomposition (TSOD) in $\mathcal{O}(n^3)$. We also desire highly regular data movement inherited in these algorithms in order to implement these efficiently on the distributed-memory MIMD multiprocessors. The algorithms are based upon "chasing" strategies for updating and downdating procedures for orthogonal decompositions.

14. SUBJECT TERMS

two-sided orthogonal decomposition, singular value decomposition, signal subspace methods, updating algorithms

15. NUMBER OF PAGES

182

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UNLIMITED

The Pennsylvania State University
APPLIED RESEARCH LABORATORY
P.O. Box 30
State College, PA 16804

MODIFYING TWO-SIDED ORTHOGONAL
DECOMPOSITIONS: ALGORITHMS, IMPLEMENTATION,
AND APPLICATIONS .

by
Peter A. Yoon

Technical Report No. TR 95-002
November 1995

L.R. Hettche, Director
Applied Research Laboratory

Abstract

Two-sided orthogonal decompositions (TSOD) have been essential tools for estimating the numerical rank of a matrix and computing various important subspaces including the range (signal) and null (noise, error) spaces. They include partial and complete singular value decompositions (SVD), the URV decomposition (URVD), and the ULV decomposition (ULVD).

Given the TSOD of an m-by-n ($m \geq n$) matrix A, it is often desirable to successively add a new row to A and to compute the TSOD of the modified matrix. This is called the *updating* problem. The opposite computation, the *downdating* problem, deletes the existing row from A, and computes the TSOD of the modified matrix. These problems of updating and downdating can be transformed into those of modifying a symmetric positive definite matrix by a rank-one matrix.

In this thesis we propose several algorithms for rank-one updates and downdates to these decompositions with strong stability properties and efficient implementations on high-performance computers. We seek algorithms which only require $\mathcal{O}(n^2)$ operations per update or downdate unlike recomputing the TSOD in $\mathcal{O}(n^3)$. We also desire highly regular data movement inherited in these algorithms in order to implement these efficiently on the distributed-memory MIMD multiprocessors. The algorithms are based upon "chasing" strategies for updating and downdating procedures for orthogonal decompositions.

In modifying the SVD and partial SVD, our algorithms separate singular values into "large" and "small" sets and then obtain an updated bidiagonal form with corresponding "large" and "small" columns. This makes more accurate update or downdate.

The algorithm can be implemented almost identically for both updating and downdating by reducing the problems to a 2 × 2 updating/downdating problem. Moreover, the bidiagonal reduction phase is highly parallelizable. A perturbation theory for modifying the SVD is also presented; it shows that the computed subspaces associated with large and small singular values are as accurate as can be expected.

An alternative to performing the singular value decomposition is to factor a matrix into $A = U \begin{pmatrix} C \\ 0 \end{pmatrix} V^T$ where U and V are orthogonal matrices and C is a lower triangular matrix indicating a separation between two subspaces by column size. These subspaces are denoted by $V = (V_1 \ V_2)$, where the columns of C are partitioned conformally into $C = (C_1 \ C_2)$ with $\|C_2\|_F \leq \epsilon$. Here ϵ is some tolerance. In recent years, this has been called the ULVD. A downdating algorithm is proposed which preserves the structure in the downdated matrix $\bar{C}$ to the extent possible. Strong stability results have been proven for these algorithms based on a new perturbation theory. When C is given as an upper triangular matrix, we have the URVD. We describe algorithms for modifying the URVD, and make comparison with our algorithms for modifying the ULVD in terms of the computed subspaces.

When downdating the ULVD, a deflation step is necessary to compute its numerical rank. We propose an improved algorithm which almost always guarantees the rank-revealing structure of the decomposition after a downdate without the deflation process. This requires some condition estimation. Moreover, one can monitor the condition of the downdating problem by tracking exact quantities of Frobenius norms of all three blocks of the lower triangular factor in the decomposition. The algorithm is also used to update the ULVD with a slight modification.

A fully parallel algorithm for modifying the SVD is also presented. We consider both cyclic and consecutive storage schemes. We will show that the latter scheme outperforms the former on a coarse-grain distributed-memory MIMD multiprocessor mainly due to high communication cost required by the former. We present the experimental results on the 32-node Connection Machine (CM-5). A speed-up of 20 and the efficiency of CPU utilization 60% are achieved for matrices of moderate size.

Our algorithms for modifying the TSOD offer a promising approach to a number of problems like the recursive total least squares, linear regression, the subspace-based methods for signal processing, image processing, and pattern recognition. These problems require a real-time solution in estimating the numerical rank of the data matrices, and orthonormal basis for the subspaces associated with large and small singular values. Our algorithms are capable of providing those answers since continual updating and downdating are required by the underlying physical model.

Table of Contents

List of Tables

List of Figures

Acknowledgments

It is impossible for me to cover all who have shaped me. First of all, I am grateful to Dr. Jesse Barlow, my adviser and friend, whose skillful hands of guidance and constant encouragement have made this dissertation possible. I am also grateful to my committee members, Drs. Leon Sibul, Mary Jane Irwin, and Hongyuan Zha, for reading the manuscript and providing their insight and helpful suggestions. In addition, I would like to express special gratitude to Dr. Sibul for his patient guidance and a delightful working environment at the Applied Research Laboratory, and to the Office of Naval Research for their generous financial support.

I would like to thank the Thinking Machine Corporation for time on CM-2, and the Northeast Parallel Architectures Center (NPAC) for time on CM-5.

I am also indebted to a special group of individuals who prayed for me throughout this writing: my parents, my pastor Sang-Kee Eun, and members of the College Fellowship of my church.

Finally, I thank Jesus Christ my Lord and Saviour for walking with me every step of this long journey, and for Emily who prayed for me and with me, and for Caleb who played with me and cheered me up with wonderful smiles.

Chapter 1

Introduction

1.1 Statement of Problem

1.1.1 Two-Sided Orthogonal Decompositions (TSOD)

The TSOD of an $m \times n$ matrix A, where $m \geq n$, can be characterized by writing them in the form

$$A = U \begin{pmatrix} M \\ 0 \end{pmatrix} V^T \qquad (1.1)$$

where $U \in \mathcal{R}^{m \times m}$ and $V \in \mathcal{R}^{n \times n}$ are orthogonal, and $M \in \mathcal{R}^{n \times n}$ has one of the following forms: diagonal, partially reduced bidiagonal, upper triangular, lower triangular.

If M is a diagonal matrix of the form,

$$M = \text{diag}\,(\sigma_1, \sigma_2, \ldots, \sigma_n) \qquad (1.2)$$

where we presume

$$\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n, \quad \|(\sigma_{k+1}, \ldots, \sigma_n)\|_2 \leq \epsilon, \quad \sigma_k \geq tol$$

and tol is the user-supplied tolerance, (1.1) is called the singular value decomposition (SVD).

If M is a partially reduced bidiagonal matrix of the form,

$$M = \begin{array}{c} \\ \begin{array}{cc} k & n-k \end{array} \\ \begin{pmatrix} B_1 & 0 \\ 0 & B_2 \end{pmatrix} \begin{array}{c} k \\ n-k \end{array} \end{array} \tag{1.3}$$

where

$$\|B_1^{-1}\|_F^{-1} > tol, \quad \|B_2\|_F \leq \epsilon,$$

and one of B_1 and B_2 is upper bidiagonal, and the other diagonal, (1.1) is called the partial singular value decomposition (partial SVD) described by Van Huffel [118]. Here, we presume they are decoupled, namely, $(k, k+1)$ entry of M is zero.

If M is an upper triangular matrix of the form,

$$M = \begin{array}{c} \\ \begin{array}{cc} k & n-k \end{array} \\ \begin{pmatrix} R & S \\ 0 & T \end{pmatrix} \begin{array}{c} k \\ n-k \end{array} \end{array} \tag{1.4}$$

where

$$\|(S^T \ T^T)\|_F \leq \epsilon, \quad \|R^{-1}\|_2^{-1} > tol,$$

and R and T are upper triangular, (1.1) is called the URV decomposition (URVD).

If M is a lower triangular matrix of the form,

$$M = \begin{array}{c} \\ \begin{array}{cc} k & n-k \end{array} \\ \begin{pmatrix} L & 0 \\ F & G \end{pmatrix} \begin{array}{c} k \\ n-k \end{array} \end{array} \tag{1.5}$$

where

$$\|(F \ G)\|_F \leq \epsilon, \quad \|L^{-1}\|_2^{-1} > tol,$$

and L and G are lower triangular, (1.1) is called the ULV decomposition (ULVD).

Here k is the numerical rank of A and $\epsilon \leq \sqrt{n-k} * tol$. We use $\|\cdot\|$ to denote the Euclidean norm and $\|\cdot\|_F$ to denote the Frobenius norm of a matrix.

The SVD has been one of the most important tools widely used in a number of fields of science and engineering for decades, mainly because it offers abundant information about the matrix in question. The SVD has many benefits. It provides orthonormal basis for important subspaces associated with the matrix including the range (signal) and null (error, noise) spaces.

The URVD and ULVD are particular cases of what Lawson and Hanson [70] called HRK decompositions. Both URVD and ULVD were introduced by Stewart [101, 102], as an alternative to the accurate but expensive SVD. Stewart also gave methods to update these decompositions in $\mathcal{O}(n^2)$ operations.

In fact, all of these decompositions provide valuable information about the data matrices. Most importantly for many applications, they provide the orthonormal basis for the range and null spaces. For instance, if V is partitioned according to

$$V = (V_1 \ V_2), \quad V_1 \in \mathcal{R}^{n \times k}, \ V_2 \in \mathcal{R}^{n \times (n-k)} \tag{1.6}$$

then it is not difficult to see that the columns of V_2 give the desired orthonormal basis for the approximate null space.

1.1.2 Modifying the TSOD

We are interested in computing the TSOD of $\bar{A}$ when the TSOD of A is known, where for *updating*,

$$\bar{A} = \begin{pmatrix} A \\ r^T \end{pmatrix}, \tag{1.7}$$

and for *downdating*,

$$A = \begin{pmatrix} r^T \\ \bar{A} \end{pmatrix}. \tag{1.8}$$

Here, we assume appending a new row to A to be the last row of $\bar{A}$, and deleting the first row of A when downdating. The downdating problem is considered more sensitive than the updating problem because small singular values of A tend to diminish after a downdate, leaving the matrix near singular, and thus can be unstable [99]. On the other hand, updating increases all its singular values. Clearly, refactoring the whole decomposition without using the TSOD of A is not practical; it requires $\mathcal{O}(n^3)$ operations to compute any TSOD.

1.2 Problem Formulation

We transform the updating/downdating problem into the rank-one modification of the symmetric eigenvalue problem. Since from (1.7) and (1.8),

$$\begin{aligned} \bar{A}^T \bar{A} &= A^T A + \rho r r^T \\ &= V M^T M V^T + \rho r r^T \\ &= V(M^T M + \rho z z^T) V^T \end{aligned}$$

where $\rho > 0$ for updating and $\rho < 0$ for downdating, and

$$z = \begin{pmatrix} x \\ y \end{pmatrix} \begin{matrix} k \\ n-k \end{matrix} \qquad (1.9)$$

Thus, the problem of modifying the TSOD of A is equivalent to the following eigenvalue problem: given a symmetric positive definite matrix $A^T A$ with known eigensystem $A^T A = V M^T M V^T$, compute the eigensystem of $M^T M + \rho z z^T$, that is, to find an orthogonal matrix $\bar{V} \in \mathcal{R}^{n \times n}$ such that

$$M^T M + \rho z z^T = \bar{V} \bar{M}^T \bar{M} \bar{V}^T. \qquad (1.10)$$

However, we do not form the explicit product $A^T A$ because of possible loss of information in forming $A^T A$. Furthermore, the eigendecompositions do not, in general, preserve the block structure. Instead, we compute orthogonal matrices $\bar{U} \in \mathcal{R}^{(n+1) \times (n+1)}$, $\bar{V} \in \mathcal{R}^{n \times n}$ such that

$$\bar{U} \begin{pmatrix} \bar{M} \\ 0 \end{pmatrix} \bar{V}^T = \begin{pmatrix} M \\ z^T \end{pmatrix} \quad \text{for updating,} \qquad (1.11)$$

and

$$\bar{U} \begin{pmatrix} z^T \bar{V} \\ \bar{M} \end{pmatrix} \bar{V}^T = \begin{pmatrix} 0 \\ M \end{pmatrix} \quad \text{for downdating,} \qquad (1.12)$$

where $\bar{M}$ is bidiagonal for the (partial) SVD, upper triangular for the URVD, and lower triangular for the ULVD.

Then $\bar{A}$ is given by

$$\bar{A} = J\widetilde{U}J^T \begin{pmatrix} \bar{M} \\ 0 \end{pmatrix} \widetilde{V}^T \tag{1.13}$$

where

$$\widetilde{V} = V\bar{V}, \quad \widetilde{U} = U\operatorname{diag}(\bar{U}, I_{m-n-1}),$$

$$J = \begin{pmatrix} \overset{m-1}{I_{m-1}} & \overset{1}{0} \end{pmatrix}, \quad \text{for updating,}$$

and

$$J = \begin{pmatrix} \overset{1}{0} & \overset{m-1}{I_{m-1}} \end{pmatrix}, \quad \text{for downdating.}$$

In theory, $\bar{M}^T \bar{M} + \rho z z^T$ remains positive semi-definite after downdating, but it is not always true in finite precision floating-point arithmetic.

1.3   Importance of the Problem

Updating and downdating are important in signal processing and statistical applications as new observations are added, and the old observations are successively deleted. They can efficiently be applied to problems which arise in a number of applications: recursive total least squares problems [23, 34], linear regression [112, 123], linear prediction [113], pattern recognition [16], system identification [60, 105], spectral estimation [17, 28, 68], adaptive beamforming [76, 77, 96], image processing/restoration [5, 7, 82], adaptive filtering [64], direction finding [2, 73], subspace-based algorithms in signal processing such as MUSIC (Multiple Signal Classification) [94, 95] and ESPRIT (Estimation

of Signal Parameters via Rotational Invariance Techniques) [91, 92], and ocean acoustic tomography [110].

1.4   Issues and Concerns

Algorithms for modifying the TSOD must have at least the following features:

Efficiency The algorithm should require as few operations as possible, for example, $\mathcal{O}(n^2)$. This feature would make it possible to implement the algorithms for applications that require a real-time processing, where continual updating/downdating of the decompositions is required.

Stability The algorithm should produce correct answers within the uncertainties of the given data. Therefore, the computed solution should be as good as our data warrants.

Parallelism It should be easy to implement the given algorithm on a parallel processor. It is desirable to develop parallel procedures which achieve the best possible load balance and minimize the communication cost, showing high efficiency and a good speed-up even for small sized problems.

1.5   Basic Approach to the Problem

Our approaches to modifying the TSOD use ideas from "chasing" algorithms [1, 93, 115, 125] and from the downdating algorithm due to Saunders [46, 85]. Chasing algorithms apply a series of Givens rotations from both sides to annihilate all components of z, reducing M to a desired form. These algorithms offer highly regular data movement, so powerful pipelining strategies can be used on parallel computers. A systolic array

implementation of a scheme with similar data movement patterns to this one (but not similar numerical properties) is implemented in [117].

The alternative to chasing algorithms for modifying the SVD is that of finding the zeroes of a particular spectral function [10, 25, 49, 53, 54, 67, 97]. However, that approach, as yet, does not allow us to separate the singular values into separate blocks in the manner discussed in Chapter 6.

1.6 Main Results

The following are the main results of this thesis.

- Blockwise procedures for modifying the TSOD which preserves the separation between subspaces associated with the "large" and "small" singular values.

- An error analysis of these procedures demonstrating that the subspaces of the modified matrix are as good as can be expected.

- Efficient parallel implementation for modifying the TSOD that incorporates clever pipelining strategies using highly regular data movement inherited from the chasing algorithms.

1.7 Review of Related Work

As mentioned in Section 1.2, problems of modifying the TSOD can be viewed as modifying the symmetric positive definite matrix followed by a rank-one matrix. Gill, Golub, Murray, and Saunders [46] considered a problem of modifying the decomposition of a matrix following a rank-one change, where they showed how to construct recurrences for the product of Givens rotations in order to modify Cholesky factor. An algorithm by Saunders was also described for downdating QR factorization, which has become

a backbone of a number of downdating algorithms including those proposed in this dissertation.

Bunch, Nielsen and Sorensen [26] studied a problem of rank-one modification of the symmetric eigendecomposition that, in turn, gave a rise to their algorithm for updating the SVD [25]. Their method was based on solving the secular equations.

Dongarra and Sorensen [38] proposed an algorithm that always computes the eigenvalues of tridiagonal matrices with high relative accuracy. However, when eigenvalues are clustered together, their algorithms had difficulties in computing numerically orthogonal eigenvectors.

An improved version was proposed by Sorensen and Tang [97], in which they incorporated simulated extended precision to overcome the difficulties of previous algorithm. However, using the simulated extended precision made the algorithm require IEEE floating-point arithmetic.

With careful rearrangement of computations in solving secular equations, Gu and Eisenstat [54] have succeeded in developing a backward stable algorithm which computes numerically orthogonal eigenvectors without using the simulated extended precision. They also observed that by using the fast multipole method of Carrier, Greengard, and Roklin [29, 30], eigenvectors can be computed in $\mathcal{O}(n^2)$ operations as compared to $\mathcal{O}(n^3)$ for the QR algorithms [47, 48]. They applied this technique further to symmetric tridiagonal eigenproblems [56], bidiagonal SVD problem [55], and the problem of downdating the SVD [57].

To this end several chasing strategies, which originated from Rutishauser [93], have been proposed for updating the SVD [27], reducing bordered band matrices [1, 52, 115], and their parallel versions [116, 117], and two-way chasing scheme by Zha [125]. Zha's algorithm improved conventional one-way chasing procedures by about a factor of 2.

These chasing schemes implemented a number of algorithms for modifying the SVD and partial SVD [1, 14], updating the URVD [101] and ULVD [102], downdating the URVD [87] and ULVD [13], refinement techniques for the URVD and ULVD [104, 100], and modifying the ULLV decomposition of two matrices [22, 75]. Parallel versions of some of these algorithms were also studied in [81, 103, 124].

1.8   Overview of the Dissertation

In the next chapter, we review some fundamental concepts from linear algebra used throughout this dissertation. After introducing notations and basic notions, we discuss briefly various types of orthogonal decompositions and their relations to the TSOD. Since we assume that the initial TSOD is given for all of our algorithms, we describe methods for computing the TSOD as well as their numerical properties. We also describe the recursive total least squares (RTLS) problems as a potential application for our algorithms.

Chapter 3 contains a detailed description of basic algorithms frequently used in the subsequent chapters. Some of algorithms include those for computing and applying a Givens rotation, various chasing algorithms, and the LINPACK [37] downdating algorithm. We also give special treatment for 2-by-2 updating/downdating procedures.

Chapter 4 discusses methods for modifying the ULVD. First, we present a detailed description of the ULVD downdating algorithm. An error analysis for this algorithm is also given to verify that the accuracy of the computed subspaces for large and small singular values is assessed. Finally, we give numerical tests of our algorithm in the context of the RTLS.

An improved algorithm for downdating the ULVD is proposed in Chapter 5. When downdating the ULVD of a matrix, a deflation step is necessary to compute its

numerical rank. We propose an efficient algorithm that almost always guarantees the rank-revealing structure of the decomposition after a downdate without the deflation process. This always requires some condition estimation. Moreover, we show how to track exact quantities of Frobenius norms of all three blocks of the lower triangular factor in the decomposition in order to monitor the condition of the downdating problem. The algorithm can also be used to update the ULVD with a slight modification.

In Chapter 6, methods for modifying the SVD and partial SVD are introduced. The main feature of these methods is the ability to separate the singular values into large and small sets and then obtain an updated bidiagonal form with corresponding large and small columns. A perturbation theory for updating and downdating the singular value decomposition is also presented.

We present a fully parallel algorithm for modifying the TSOD in Chapter 7. Both cyclic and consecutive storage schemes are considered in parallel implementation. We show that the latter scheme outperforms the former on a coarse-grain distributed-memory MIMD multiprocessor. We give the experimental results on the 32-node Connection Machine (CM-5).

Finally, we give our conclusion and propose future work in Chapter 8.

Chapter 2

Background

In this chapter we present briefly important concepts from linear algebra frequently used throughout the dissertation. For detailed description of each subject, refer to [50, 58, 89, 98, 106, 121, 122].

2.1 Notation and Basic Notions

We use the following notations throughout this dissertation.

$\mathcal{R}$ Set of real numbers denoted by lower case Greek or lower case *italic* if there is no confusion

$\mathcal{R}^n$ Set of real $n-$vectors denoted by lower case *italic*

$\mathcal{R}^{m \times n}$ Set of real m-by-n matrices denoted by upper case *ITALIC* or upper case Greek letters

I_n n-by-n identity matrix, that is, $I = (e_1, \ldots, e_n)$, where $e_k = (\overbrace{0, \ldots, 0}^{k-1}, 1, \overbrace{0, \ldots, 0}^{n-k})^T$

O_n n-by-n zero matrix

A^T Transpose of A

A^H Complex conjugate of A^T

A^{-1} Inverse of $A \in \mathcal{R}^{n \times n}$, that is, $A^{-1}A = AA^{-1} = I_n$

$\text{span}\{a_1, \ldots, a_n\}$ $\{\sum_{i=1}^n \beta_i a_i, \beta_i \in \mathcal{R}\}$

$\text{range}(A)$ $\{y \in \mathcal{R}^m : y = Ax, x \in \mathcal{R}^n\} = \text{span}\{a_1, \ldots, a_n\}$

$\text{null}(A)$ $\{x \in \mathcal{R}^m : Ax = 0\}$

rank(A)   dim(range(A))

$\lambda(A)$ Set of eigenvalues of A, that is, $\{\lambda \in \mathcal{R} : Ax = \lambda x,\ 0 \neq x \in \mathcal{R}^n\}$

$\sigma(A)$ Set of singular values of A

$\sigma_i(A)$ i-th singular value of A in nondecreasing order, that is, $\sigma_i(A) = \sqrt{\lambda_i(A^T A)}$

$\sigma_{\max}(A)$ Largest singular value of A

$\sigma_{\min}(A)$ Smallest singular value of A

$\kappa(A)$ condition number of A, $\kappa(A) = \sigma_{\max}(A)/\sigma_{\min}(A)$

$\mathcal{O}(\cdot)$ $g(n) = \mathcal{O}(f(n))$ if there exist constants c and N, such that, for all $n \geq N$, we have $g(n) \leq cf(n)$ [3]

flop   A floating point operation, that is, the amount of work associated with an addition, a multiplication, or a square root

μ Machine unit

sign(x) $x/|x|$ if $x \neq 0$; 1 if $x = 0$

DEFINITION 2.1 (VECTOR NORMS). *Let $x \in \mathcal{R}^n$. Then*

$$\|x\|_1 = \sum_{i=1}^{n} |x_i|, \quad \|x\|_\infty = \max_i |x_i|, \quad \|x\|_2 = \left(\sum_{i=1}^{n} x_i^2\right)^{\frac{1}{2}}$$

DEFINITION 2.2 (MATRIX NORMS). *Let $A \in \mathcal{R}^{m \times n}$. Then*

$$\|A\|_p = \max_{x \neq 0} \frac{\|Ax\|_p}{\|x\|_p}, \quad \|A\|_F = \left(\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2\right)^{\frac{1}{2}}$$

where $p = 1, 2, \infty$.

We use $\| \cdot \|$ to denote Euclidean norm $\| \cdot \|_2$, and $\| \cdot \|_F$ to denote Frobenius norm. It can be easily shown that $\sigma_{\max}(A) = \|A\|$.

Next, we devise a notion of distance between subspaces.

DEFINITION 2.3 ([50, P.77]). *Let $W = (W_1 \quad W_2)$ and $Z = (Z_1 \quad Z_2)$ be orthogonal matrices where $W_1, Z_1 \in \mathcal{R}^{n \times k}$ and $W_2, Z_2 \in \mathcal{R}^{n \times (n-k)}$. If $S_1 = \text{range}(W_1)$ and $S_2 = \text{range}(Z_1)$, then $\text{dist}(S_1, S_2) = \|W_2^T Z_1\| = \sqrt{1 - \sigma_{\min}^2(W_1^T Z_1)}$.*

Thus, if $\sin(\theta) = \text{dist}(S_1, S_2)$ for some θ, then θ is the largest angle between the two subspaces.

DEFINITION 2.4. *A matrix $A \in \mathcal{R}^{m \times n}$ is*

diagonal	*if $a_{ij} = 0$, $\quad i \neq j$;*
tridiagonal	*if $a_{ij} = 0$, $\quad |i - j| > 1$;*
upper bidiagonal	*if $a_{ij} = 0$, $\quad i > j$ or $j > i + 1$;*
lower bidiagonal	*if $a_{ij} = 0$, $\quad i < j$ or $j < i - 1$;*
upper triangular	*if $a_{ij} = 0$, $\quad i > j$;*
lower triangular	*if $a_{ij} = 0$, $\quad i < j$;*
upper Hessenberg	*if $a_{ij} = 0$, $\quad i > j + 1$;*
lower Hessenberg	*if $a_{ij} = 0$, $\quad i < j - 1$.*

DEFINITION 2.5. *A matrix $A \in \mathcal{R}^{n \times n}$ is*

symmetric	*if $A^T = A$;*
positive definite	*if $x^T A^T x > 0$, $0 \neq x \in \mathcal{R}^n$;*
positive semi-definite	*if $x^T A^T x \geq 0$, $0 \neq x \in \mathcal{R}^n$;*
orthogonal	*if $A^T A = A A^T = I_n$;*
permutation	*if $A = (e_{s_1}, \ldots, e_{s_n})$ where $(s_1, \ldots, s_n)$ is a permutation of $(1, \ldots, n)$.*

$\| \cdot \|$ and $\| \cdot \|_F$ are *orthogonally invariant* norms, that is, for any $x \in \mathcal{R}^n$, $A \in \mathcal{R}^{m \times n}$, and orthogonal matrices, $Q \in \mathcal{R}^{m \times m}$ and $Z \in \mathcal{R}^{n \times n}$, we have

$$\|Qx\| = \|x\|, \quad \|QAZ\| = \|A\|, \quad \|QAZ\|_F = \|A\|_F.$$

In theory, if $\text{rank}(A) = k$ for $A \in \mathcal{R}^{m \times n}$, then we have

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_k > 0, \quad \sigma_{k+1} = \cdots = \sigma_n = 0$$

where σ_i are singular values of A defined in (1.2). However, in practice, σ_{k+1} is not exactly equal to zero, but $\sigma_{k+1} = \mathcal{O}(\mu)$, where μ is the machine unit. Therefore, we define the numerical $\epsilon - rank$ to be the largest integer k such that

$$\sigma_k > \epsilon$$

that is,

$$\sigma_k \gg \sigma_{k+1}$$

meaning that there exists an obvious gap between the singular values. A number of signal identification problems assume a significant gap in the singular value spectrum of the data matrix. Thus, approximate range and null spaces associated with the large and small singular values can be defined accordingly.

2.2 Stability of Algorithms

Let $\mathcal{F}(x)$ be a function of the input data x.

DEFINITION 2.6. *An algorithm for computing $\mathcal{F}(x)$ is* **backward stable** *if the computed solution $\widehat{\mathcal{F}}(x)$ is the* **exact** *solution of a slightly perturbed problem with data $\widehat{x}$.*

This definition is similar to that of Bunch [24]. Backward stable algorithms are very satisfactory although all of the algorithms proposed in this thesis for modifying the TSOD are not backward stable. They are *mixed stable* as defined in the following sense:

DEFINITION 2.7. *An algorithm for computing $\mathcal{F}(x)$ is* **mixed stable** *if the computed solution $\widehat{\mathcal{F}}(x)$ is* **close** *to the solution of a slightly perturbed problem with data $\widehat{x}$.*

Definition 2.7 is used in the context of modifying orthogonal decompositions [21, 83, 99], and downdating least squares solutions [20]. We will also use this definition of stability when we analyze our algorithms in the subsequent chapters.

2.3 Model of Computation

The machine unit μ is the smallest number which satisfies

$$|fl(a \operatorname{op} b) - (a \operatorname{op} b)| \leq \mu \, |a \operatorname{op} b| \tag{2.1}$$

where op is one of the four arithmetic operations, $+, -, \times, \div$, and $fl(a \operatorname{op} b)$ is the floating-point representation of the exact result $a \operatorname{op} b$. We take the usual model of arithmetic, assuming underflow/overflow does not occur,

$$fl(a \operatorname{op} b) = (a \operatorname{op} b)(1 + \xi), \quad |\xi| \leq \mu. \tag{2.2}$$

Most computers take this model except for those without guard digits such as Cray for which

$$fl(a \operatorname{op} b) = a(1 + \xi_1) \operatorname{op} b(1 + \xi_2), \quad |\xi_1|, |\xi_2| \leq \mu$$

With this model we may obtain unsatisfactory results in addition and subtraction.

We also require for our model

$$fl(\sqrt{x}) = \sqrt{x}(1+\xi), \quad |\xi| \le \mu, \quad x > 0. \tag{2.3}$$

2.4 Orthogonal Factorizations

Orthogonal decompositions play an important role in a number of problems in matrix computations such as least squares and eigenvalue problems. In this section we review two special orthogonal transformations: Householder reflections and Givens rotations, and a family of orthogonal decompositions that can be computed by a series of application of these transformations.

2.4.1 Householder Transformation

A Householder transformation (reflection) of order n takes the form

$$H = I_n - 2vv^T/v^Tv \tag{2.4}$$

where v is called a Householder vector. It is easy to verify that H is symmetric and orthogonal. The Householder transformation can be used to annihilate a number of components of a vector. For example, let $x \in \mathcal{R}^n$. If we choose v such that

$$v_i = 0, \quad i = 1, \ldots, k-1$$

$$v_k = x_k - s,$$

$$v_i = x_i, \quad i = k+1, \ldots, n$$

where

$$s = -\text{sign}(x_k)\left(\Sigma_{i=k}^{n} x_i^2\right)^{\frac{1}{2}},$$

then we obtain

$$Hx = (x_1, x_2, \ldots, x_{k-1}, s, 0, \ldots, 0)^T.$$

Note that we always choose s so that s and x_k can have opposite sign to prevent the loss of precision in the computation.

2.4.2 Givens Transformation

When one wishes to zero elements more selectively, Givens transformations (rotations) do the best job. A Givens rotation takes the form

$$J(i,j,\theta) = \begin{pmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & c & \cdots & s & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & -s & \cdots & c & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{pmatrix} \begin{matrix} \\ \\ i \\ \\ j \\ \\ \\ \end{matrix}$$

where $c = \cos(\theta)$ and $s = \sin(\theta)$ for some θ.

Given a vector $x \in \mathcal{R}^n$, if we choose

$$c = \frac{x_i}{\rho}, \quad s = \frac{-x_j}{\rho}, \quad \rho = \sqrt{x_i^2 + x_j^2},$$

then we obtain

$$J(i,j,\theta)^T x = (x_1, \ldots, x_{i-1}, \rho, x_{i+1}, \ldots, x_{k-1}, 0, x_{k+1}, \ldots, x_n)^T.$$

Algorithm 3.2 described in the next chapter computes c and s without causing overflow and underflow in computing ρ.

2.4.3 Rank Revealing QR Factorization

The QR factorization of an m-by-n matrix A is given by

$$A = Q \begin{pmatrix} R \\ 0 \end{pmatrix} \begin{matrix} n \\ m-n \end{matrix} \tag{2.5}$$

$$Q = (Q_1 \ \ Q_2), \quad Q_1 \in \mathcal{R}^{m \times n}, \ Q_2 \in \mathcal{R}^{m \times (m-n)}$$

where $Q \in \mathcal{R}^{m \times m}$ is orthogonal and $R \in \mathcal{R}^{n \times n}$ is upper triangular.

If A has full column rank, the columns of Q_1 spans range(A). However, when A is near rank-deficient, and computing orthonormal bases for range(A) and null(A) is of interest, the factorization of the form (2.5) is obviously inadequate.

It can be shown that with a careful rearrangement of columns of A with some pivoting, one can produce another QR factorization which *reveals* the numerical rank of A. A rank-revealing QR (RRQR) factorization of $A \in \mathcal{R}^{m \times n}$ is any decomposition

$$A\Pi = Q \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \\ 0 & 0 \end{pmatrix} \begin{matrix} k \\ n-k \\ m-n \end{matrix} \tag{2.6}$$

$$Q = (Q_1 \quad Q_2), \quad Q_1 \in \mathcal{R}^{m \times n}, \ Q_2 \in \mathcal{R}^{m \times (m-n)}$$

where R_{11} and R_{22} are upper triangular, R_{11} is well-conditioned, $\|R_{11}^{-1}\|^{-1} \approx \sigma_k(A)$, $\|R_{22}\| \approx \sigma_{k+1}(A)$, and Π is a permutation matrix. Here, k is the numerical rank of A. Note that $\|R_{12}\|$ is not necessarily small compared to $\|R_{11}\|$. Thus, the factorization does not produce explicitly the approximate null space of A.

Chan [31], however, shows that RRQR factorization may produce the approximate null space for A with high accuracy when there is a large gap in the singular value spectrum of A although that may not be a practical assumption in some applications. His algorithm incorporates several techniques such as an efficient condition estimation and column pivoting. His algorithm also guarantees an RRQR factorization for a high-rank problem.

Using similar ideas, Foster [44] independently developed a stable algorithm for determining the numerical rank of a matrix without requiring column interchanges. Hong and Pan [62] recently showed that the permutation Π always exists, and gave a method for constructing such permutation. However, because of the high cost required by the procedure, it has more theoretical than practical value.

2.4.4 Complete Orthogonal Decompositions

It turns out that with an appropriate choice of a general orthogonal matrix $Z \in \mathcal{R}^{n \times n}$ (considering permutation matrices as a special class of orthogonal matrices), one

can reduce R in (2.6) even further so that $\|R_{12}\|$ becomes small as well:

$$
AZ = Q \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \\ 0 & 0 \end{pmatrix} \begin{matrix} k \\ n-k \\ m-n \end{matrix} \quad ,
\tag{2.7}
$$

with column labels k and $n-k$ over R_{11} and R_{12}.

$$
Z = (Z_1 \; Z_2), \quad Z_1 \in \mathcal{R}^{n \times k}, \; Z_2 \in \mathcal{R}^{n \times (n-k)}
$$

where $\|R_{11}^{-1}\|^{-1} \approx \sigma_k(A)$, and

$$
\left\| \begin{pmatrix} R_{12} \\ R_{22} \end{pmatrix} \right\| \approx \sigma_{k+1}(A).
$$

Then, it is easy to see that columns of Z_2 provides the orthonormal basis for the approximate null space.

2.5 Computing the TSOD

2.5.1 Computing the Singular Value Decomposition

A standard way of computing the SVD of $A \in \mathcal{R}^{m \times n}$ involves two steps: bidiagonalization and computation of the SVD of resulting bidiagonal matrix. The first step requires to find products of Householder transformations, $U = U_1 \cdots U_{n-1} \in \mathcal{R}^{m \times m}$ and $V = V_1 \cdots V_{n-1} \in \mathcal{R}^{n \times n}$ such that

$$
U^T A V = \begin{pmatrix} B \\ 0 \end{pmatrix} \begin{matrix} n \\ m-n \end{matrix}
$$

where B is upper bidiagonal. Then, we compute orthogonal matrices $P, Q \in \mathcal{R}^{n \times n}$ such that

$$B = P \Sigma Q^T, \quad \Sigma = \mathrm{diag}(\sigma_1, \ldots, \sigma_n).$$

Thus we obtain

$$A = X \begin{pmatrix} \Sigma \\ 0 \end{pmatrix} Y^T$$

where $X = U \, \mathrm{diag}(P, I_{m-n})$ and $Y = VQ$.

The whole process requires $\mathcal{O}(m^2 n)$ flops. The second phase of computing the SVD of bidiagonal matrix can be done by QR-iteration [47, 48] which is implemented in the LINPACK [37]. But the singular values computed by this method differ from the true singular values by at most $p(n) \cdot \mu \cdot \sigma_{\max}(A)$, where $p(n)$ is a moderately growing function of n. Thus, large singular values are computed with high relative accuracy, but small ones are not generally accurate.

Demmel and Kahan [35] developed an algorithm for computing all the singular values of a bidiagonal matrix to maximal relative accuracy independent of their magnitudes. Their algorithm implemented in LAPACK [6], is essentially the QR-iteration incorporated with a "zero-shift", which is often faster than the standard algorithm implemented in the LINPACK.

Fernando and Parlett [40] simplified the zero-shift bidiagonal algorithm by Demmel and Kahan even further by replacing a zero-shift QR step with two steps of LR iteration that implement the **qd** algorithm. We describe the **qd** algorithm in the next chapter.

Other methods for computing all the singular values with high relative accuracy of a bidiagonal matrix include bisection [15], Rayleigh quotient iteration [88]. But they are

not competitive in speed with the zero-shift bidiagonal algorithm and the **qd** algorithm, although they are probably the most parallelizable algorithms for this problem.

It is a well-known fact that reducing a dense matrix into bidiagonal form can introduce large relative errors in its singular values. The Jacobi method for computing the SVD of a dense matrix is much slower but more accurate than any algorithms that first bidiagonalize the matrix [36]. In this iterative algorithm, a series of Givens rotations are applied to pairs of rows and columns to reduce off-diagonal entries. With a clever ordering [74], the algorithm can be implemented in parallel, being competitive in speed with other methods.

2.5.2 Computing the ULV and URV Decompositions

The ULVD of $A \in \mathcal{R}^{m \times n}$ can be obtained by computing its QL factorization

$$A = Q \begin{pmatrix} L \\ 0 \end{pmatrix} \begin{matrix} n \\ m-n \end{matrix}$$

where $Q \in \mathcal{R}^{m \times m}$ is orthogonal and $L \in \mathcal{R}^{n \times n}$ lower triangular, followed by computing the ULVD of L using the deflation technique described in [100, 102]. First, we estimate an approximate left singular vector u_n of unit norm of L which corresponds to $\sigma_n(L)$ using some condition estimator. A survey of popular condition estimators is given in [61]. Then, we compute an orthogonal matrix $\bar{Q} \in \mathcal{R}^{n \times n}$ such that $u_n^T \bar{Q} = e_n^T$ and an orthogonal matrix $\bar{P} \in \mathcal{R}^{n \times n}$, such that

$$\sigma_n(L) = \|u_n^T L\| = \|(u_n^T \bar{Q})(\bar{Q}^T L \bar{P})\| = \|e_n^T(\bar{Q}^T L \bar{P})\|,$$

which is the size of the last row of L. Here, $\bar{P}$ is applied to restore $\bar{Q}^T L$ into the lower triangular form. We repeat this deflation process until all the small rows of L appear in the decomposition yielding the ULVD of A of the form (1.5).

Similarly, the URVD of A can be obtained by computing its QR factorization of the form (2.5) followed by computing the URVD of R using the deflation steps. This time we estimate an approximate right singular vector v_n of unit norm of R which corresponds to $\sigma_n(R)$, and then compute an orthogonal matrix $\bar{Q} \in \mathcal{R}^{n \times n}$ such that $\bar{Q}^T v_n = e_n$ and an orthogonal matrix $\bar{P} \in \mathcal{R}^{n \times n}$, so that

$$\sigma_n(R) = \|Rv_n\| = \|(\bar{P}^T R\bar{Q})(\bar{Q}^T v_n)\| = \|(\bar{P}^T R\bar{Q})e_n\|$$

which is the size of the last column of R. Here, $\bar{P}$ is applied to restore $R\bar{Q}$ into the upper triangular form. We repeat this deflation process until all the small columns of R appear in the decomposition yielding the URVD of A of the form (1.4).

2.6   Subspaces from the URV and ULV Decompositions

In this section we present error bounds for accessing the accuracy of subspaces computed by the TSOD, particularly, the ULVD and URVD. The discussion that follows is largely abridged from that of Fierro and Bunch [41, 42].

Let the SVD of $A \in \mathcal{R}^{m \times n}$, $m \geq n$ be

$$A = U \begin{pmatrix} \Sigma \\ 0 \end{pmatrix} V^T, \tag{2.8}$$

where

$$
\begin{array}{ccc} k & n-k & m-n \end{array} \qquad \begin{array}{cc} k & n-k \end{array}
$$

$$
U = \begin{pmatrix} U_1 & U_2 & U_3 \end{pmatrix}, \quad V = \begin{pmatrix} V_1 & V_2 \end{pmatrix},
$$

and Σ has the form (1.2), and let the URVD of A be

$$
A = U_R \begin{pmatrix} C_R \\ 0 \end{pmatrix} V_L^T, \tag{2.9}
$$

where

$$
\begin{array}{ccc} k & n-k & m-n \end{array} \qquad \begin{array}{cc} k & n-k \end{array}
$$

$$
U_R = \begin{pmatrix} U_{R1} & U_{R2} & U_{R3} \end{pmatrix}, \quad V_L = \begin{pmatrix} V_{R1} & V_{R2} \end{pmatrix},
$$

and C_R has the form (1.4), and let the ULVD of A be

$$
A = U_L \begin{pmatrix} C_L \\ 0 \end{pmatrix} V_L^T, \tag{2.10}
$$

where

$$
\begin{array}{ccc} k & n-k & m-n \end{array} \qquad \begin{array}{cc} k & n-k \end{array}
$$

$$
U_L = \begin{pmatrix} U_{L1} & U_{L2} & U_{L3} \end{pmatrix}, \quad V_L = \begin{pmatrix} V_{L1} & V_{L2} \end{pmatrix},
$$

and C_L has the form (1.5). Here, U, U_R, U_L, V, V_R, and V_L are orthogonal matrices.

Then we have the following bounds on subspaces computed by the URVD and ULVD, which are associated with the large and small singular values.

THEOREM 2.1 ([42]). *Let* $A \in \mathcal{R}^{m \times n}$ *have the SVD of the form (2.8) and the URVD of the form (2.9). Suppose* $\sigma_{\min}(R) > \|T\|$. *Then*

$$\frac{\|S\|}{\|R\| + \|T\|} \leq \|V_1^T V_{R2}\| \leq \frac{\sigma_{\min}(R) \|S\|}{\sigma_{\min}^2(R) - \|T\|^2} \tag{2.11}$$

$$\|U_2^T U_{R1}\| \leq \frac{\|S\| \|T\|}{\sigma_{\min}^2(R) - \|T\|^2} \tag{2.12}$$

THEOREM 2.2 ([42]). *Let* $A \in \mathcal{R}^{m \times n}$ *have the SVD of the form (2.8) and the ULVD of the form (2.10). Suppose* $\sigma_{\min}(L) > \|G\|$. *Then*

$$\|V_1^T V_{L2}\| \leq \frac{\|F\| \|G\|}{\sigma_{\min}^2(L) - \|G\|^2} \tag{2.13}$$

$$\frac{\|F\|}{\|L\| + \|G\|} \leq \|U_2^T U_{L1}\| \leq \frac{\|F\|}{\sigma_{\min}(L) - \|G\|} \tag{2.14}$$

From these theorems we immediately realize that the quality of subspaces computed by the URVD and ULVD do not depend on the gap in the singular value spectrum, which is required by the RRQR decomposition. We also observe that by keeping $\|S\|$ and $\|F\|$ small, we can obtain highly accurate subspaces. Several ways of achieving this are described in [41, 42, 104]. It can be also argued that the ULVD computes the numerical null spaces more accurately than the URVD, whereas the URVD yields a better estimate of the numerical range.

2.7 Total Least Squares Problem

2.7.1 Problem Formulation

The classical linear least squares (LS) problem is given by

$$\min_{x \in \mathcal{R}^n} \|Ax - b\|, \quad A \in \mathcal{R}^{m \times n}, \, b \in \mathcal{R}^m \tag{2.15}$$

or equivalently,

$$\min_{b+e \in \text{range}(A)} \|e\|, \quad e \in \mathcal{R}^m \tag{2.16}$$

where A is data matrix whose rows contain measurements from the model under consideration, and b is the observation vector. Here, we presume A is free of error and b is subject to error. However, it is unrealistic to take error-free measurements from the model.

The total least squares (TLS) problem assumes that there are errors in the data matrix A as well as the observation vector b. The TLS problem has the formulation

$$\min_{b+e \in \text{range}(A+E)} \|(E \, e)\|_F, \tag{2.17}$$

which is analogous to (2.16).

The errors-in-variables problems have a long history in statistical literature. In the field of numerical analysis, this problem was first introduced by Golub and Van Loan [51] and then studied extensively by Van Huffel et al. [114, 118]. If e_{LS} and $(E_{TLS} \, e_{TLS})$ are the LS and TLS corrections to (2.16) and (2.17), respectively, it can be shown that

$$\|(E_{TLS} \, e_{TLS})\|_F \leq \|e_{LS}\|$$

2.7.2 Basic Solution

The TSOD gives an elegant way of solving the TLS problem. Suppose the TSOD of $(A \ b) \in \mathcal{R}^{m \times (n+1)}$ is given by

$$(A \ b) = U \begin{pmatrix} M \\ 0 \end{pmatrix} V^T, \quad V = \begin{pmatrix} \overset{k}{V_1} & \overset{n-k+1}{V_2} \end{pmatrix}$$

where $U \in \mathcal{R}^{m \times m}$ and $V \in \mathcal{R}^{(n+1) \times (n+1)}$ are orthogonal, and $M \in \mathcal{R}^{(n+1) \times (n+1)}$ has one of the forms (1.2)–(1.5). Then $V_2 = (v_{k+1}, \ldots, v_{n+1})$ is a basis of the noise subspace of $(A \ b)$, and the minimum norm TLS solution is given by computing a Householder transformation $H \in \mathcal{R}^{(n+1) \times (n+1)}$ such that

$$V_2 H = \begin{pmatrix} Y & d \\ 0 & \delta \end{pmatrix} \begin{matrix} n \\ 1 \end{matrix} . \tag{2.18}$$

If $\delta \neq 0$, the TLS solution $\hat{x}$ is given by

$$\hat{x} = -d/\delta. \tag{2.19}$$

See [118] for details. In particular, Golub and Van Loan formulated the solution based on the SVD. Van Huffel and Zha [119] also formulated the solution to the TLS problems based on the URVD and the ULVD without the explicit computation of the approximate null space basis V_2.

2.7.3 Recursive Total Least Squares

When one does not have complete knowledge of the data in a given model, recursive procedures make it possible to achieve satisfactory results. The algorithm is given, to begin with, incomplete knowledge about the environment, and modifies the processing model in an adaptive fashion as data are received sequentially.

In recursive TLS problem, it is required to append a new row to the data matrix A and an observation to b, and the new information must be incorporated into the TLS solution. It is also desirable to delete the oldest observation from the existing data.

The updating and downdating problems of the form (1.7)–(1.8) are associated with the sliding window method [4, 19, 32]. At each step of the sliding window method with the window size s, an $s \times n$ data matrix is constructed from an $m \times n$ observation matrix A by adding a new row to the data matrix in the previous window and deleting the oldest row from it. In step j, the row $s + j$ of the observation matrix is added and the row j is deleted, giving the data matrix A_j:

$$
\cdots, A_j = \begin{pmatrix} a_{j-s+1}^T \\ a_{j-s+2}^T \\ \vdots \\ a_{j-1}^T \\ a_j^T \end{pmatrix}, \quad A_{j+1} = \begin{pmatrix} a_{j-s+2}^T \\ a_{j-s+3}^T \\ \vdots \\ a_j^T \\ a_{j+1}^T \end{pmatrix}, \cdots
$$

An alternative approach is to use an exponential forgetting factor β $(0 < \beta \leq 1)$ [59]. In this approach the modified matrix in (1.7)–(1.8) is given by

$$\bar{A} = \begin{pmatrix} \beta A \\ r^T \end{pmatrix}. \qquad (2.20)$$

Here, the effect of old observation diminishes exponentially as continuous updating is required. However, the explicit removal of the observation, as in the sliding window method, makes it simple to estimate the rank of modified matrix, that is, it remains the same or increases by one for updating, or decreases by one for downdating. Thus, an indefinite number of condition estimation steps is not necessary for rank detection as done in [101, 102].

Chapter 3

Basic Algorithms

This chapter contains detailed description of basic algorithms frequently used in the subsequent chapters. Throughout the dissertation we follow the convention of the MATLAB [78] in describing the algorithms. Some of the algorithms include those for computing and applying a Givens rotation, various chasing algorithms, and the LIN-PACK [37] downdating algorithm. We also give a special treatment for 2-by-2 updating/downdating procedures.

3.1 Givens Rotations

This section contains simple routines for constructing and applying Givens rotations described in Section 2.4.2.

ALGORITHM 3.1. This function computes complex *abs*olute value of $x + iy$, that is, $t = \sqrt{x^2 + y^2}$.

function $t = \mathbf{cabs}(x, y)$

    if $|x| \geq |y|$ **then**

        $r \leftarrow y/x; t \leftarrow |x| * \sqrt{1 + r^2}$

    else

        $r \leftarrow x/y; t \leftarrow |y| * \sqrt{1 + r^2}$

    endif

end

ALGORITHM 3.2. This function computes $c = \cos(\theta)$ and $s = \sin(\theta)$ for some θ such that

$$\begin{pmatrix} c & s \\ -s & c \end{pmatrix}^T \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} \mathbf{cabs}(a,b) \\ 0 \end{pmatrix}.$$

procedure formrot(a, b, c, s)

    if $|a| \geq |b|$ **then**

        $r \leftarrow b/a;\ factor \leftarrow \sqrt{1 + r^2}$

        $a \leftarrow |a| * factor;\ c \leftarrow 1/factor;\ s \leftarrow r * c$

    else

        $r \leftarrow a/b;\ factor \leftarrow \sqrt{1 + r^2}$

        $a \leftarrow |b| * factor;\ s \leftarrow 1/factor;\ c \leftarrow r * s$

    endif

end

ALGORITHM 3.3. This procedure applies the plane rotation defined by c and s to rotate $(x^T, y^T)^T$.

procedure applyrot(x, y, c, s, n)

    $temp \leftarrow c * x(1{:}n) + s * y(1{:}n)$

    $y(1{:}n) \leftarrow -s * x(1{:}n) + c * y(1{:}n)$

    $x(1{:}n) \leftarrow temp$

end

3.2 Chasing Algorithms

3.2.1 A Chasing Routine for a Bidiagonal Matrix

The following routine, for a given vector z and a diagonal matrix B, finds orthogonal matrices $\bar{U}$ and $\bar{V}$ such that

$$\bar{B} = \bar{U}^T B \bar{V}, \quad \bar{V}^T z = \rho e_1, \quad \rho = \|z\| \tag{3.1}$$

where $\bar{B}$ is lower bidiagonal.

Fig. 3.1 illustrates the reduction steps. The right arrows $\overset{\rightarrow}{\rightarrow}$ denote rotations from the left on two particular rows, whereas the downarrows $\downarrow\downarrow$ denote rotations from the right on two particular columns. Here z denotes elements in the vector z, b denotes elements in B, and $\hat{z}$ or $\hat{b}$ denotes an element about to be zeroed out. We now formally present the algorithm below.

ALGORITHM 3.4 (CHASING ALGORITHM FOR BIDIAGONAL REDUCTION). Given a diagonal matrix, $B = \text{diag}(\gamma(1\!:\!n))$, and a vector to be reduced, $z(1\!:\!n)$, the following chasing scheme produces a lower bidiagonal matrix $\bar{B}$ such that $\bar{B} = \text{bidiag}(\gamma(1\!:\!n), \phi(1\!:\!n-1))$ and satisfies (3.1).

procedure forchase(γ, ϕ, z, n)

    formrot $(z(n-1), z(n), cn, sn)$

    $e \leftarrow -sn * \gamma(n-1); \quad \gamma(n-1) \leftarrow cn * \gamma(n-1)$

    $\phi(n-1) \leftarrow sn * \gamma(n); \gamma(n) \leftarrow cn * \gamma(n)$

    formrot $(\gamma(n), e, cn, sn)$

    applyrot $(cn, sn, \phi(n-1), \gamma(n-1), 1)$

    for $i \leftarrow n-2, \ldots, 1$

        formrot $(z(i), z(i+1), cn, sn)$

$$
\begin{array}{c} {\scriptstyle\downarrow\ \ \downarrow} \\ \begin{pmatrix} z & z & z & \hat{z} \\ b & & & \\ & b & & \\ & & b & \\ & & & b \end{pmatrix} \end{array}
\begin{array}{c}\rightarrow\\\rightarrow\end{array}
\begin{pmatrix} z & z & z & 0 \\ b & & & \\ & b & & \\ & & b & \hat{b} \\ & & b & b \end{pmatrix}
\begin{array}{c} {\scriptstyle\downarrow\ \ \downarrow} \\ \begin{pmatrix} z & z & \hat{z} & 0 \\ b & & & \\ & b & & \\ & & b & \\ & & b & b \end{pmatrix} \end{array}
\begin{array}{c}\rightarrow\\\rightarrow\end{array}
\begin{pmatrix} z & z & 0 & 0 \\ b & & & \\ & & b & \hat{b} \\ & & b & b \\ & & b & b \end{pmatrix}
$$

$$
\begin{array}{c}\rightarrow\\\rightarrow\end{array}
\begin{pmatrix} z & z & 0 & 0 \\ b & & & \\ & & b & \\ & & b & b \\ & \hat{b} & b & b \end{pmatrix}
\begin{array}{c} {\scriptstyle\downarrow\ \ \downarrow} \\ \begin{pmatrix} z & z & 0 & 0 \\ b & & & \\ & & b & \\ & & b & b & \hat{b} \\ & & b & b \end{pmatrix} \end{array}
\begin{array}{c} {\scriptstyle\downarrow\ \ \downarrow} \\ \begin{pmatrix} z & \hat{z} & 0 & 0 \\ b & & & \\ & & b & \\ & & b & b \\ & & b & b \end{pmatrix} \end{array}
\begin{array}{c}\rightarrow\\\rightarrow\end{array}
\begin{pmatrix} z & 0 & 0 & 0 \\ b & \hat{b} & & \\ b & b & & \\ & b & b & b \\ & & b & b \end{pmatrix}
$$

$$
\begin{array}{c}\rightarrow\\\rightarrow\end{array}
\begin{pmatrix} z & 0 & 0 & 0 \\ b & & & \\ b & b & & \\ \hat{b} & b & b & \\ & & b & b \end{pmatrix}
\begin{pmatrix} z & 0 & 0 & 0 \\ b & & & \\ b & b & \hat{b} & \\ & b & b & \\ & b & b & \end{pmatrix}
\begin{array}{c} {\scriptstyle\downarrow\ \ \downarrow} \\ \begin{pmatrix} z & 0 & 0 & 0 \\ b & & & \\ b & b & & \\ & b & b & \\ & \hat{b} & b & b \end{pmatrix} \end{array}
\begin{array}{c} {\scriptstyle\downarrow\ \ \downarrow} \\ \begin{pmatrix} z & 0 & 0 & 0 \\ b & & & \\ b & b & & \\ & b & b & \hat{b} \\ & b & b & \end{pmatrix} \end{array}
$$

$$
\begin{pmatrix} z & 0 & 0 & 0 \\ b & & & \\ b & b & & \\ & b & b & \\ & & b & b \end{pmatrix}
$$

Fig. 3.1. Forward Chasing Procedure for the Bidiagonal Reduction

$$e \leftarrow -sn * \gamma(i); \gamma(i) \leftarrow cn * \gamma(i)$$

$$\phi(i) \leftarrow sn * \gamma(i+1); \gamma(i+1) \leftarrow cn * \gamma(i+1)$$

$$d \leftarrow cn * \phi(i+1); \phi(i+1) \leftarrow sn * \phi(i+1)$$

formrot $(\gamma(i), e, cn, sn)$

applyrot $(cn, sn, \phi(i), \gamma(i+1), 1)$

applyrot $(cn, sn, d, \phi(i+1), 1)$

for $j \leftarrow i, \ldots, n-3$

    formrot $(\phi(j), d, cn, sn)$

    applyrot $(cn, sn, \gamma(j+1), \phi(j+1), 1)$

    $e \leftarrow sn * \gamma(j+2); \gamma(j+2) \leftarrow cn * \gamma(j+2)$

    formrot $(\phi(j), d, cn, sn)$

    applyrot $(cn, sn, \gamma(j+1), \phi(j+1), 1)$

    $d \leftarrow sn * \phi(j+2); \phi(j+2) \leftarrow cn * \phi(j+2)$

endfor

formrot $(\phi(n-2), d, cn, sn)$

applyrot $(cn, sn, \gamma(n-1), \phi(n-1), 1)$

$e \leftarrow sn * \gamma(n); \gamma(n) \leftarrow cn * \gamma(n)$

formrot $(\gamma(n-1), e, cn, sn)$

applyrot $(cn, sn, \phi(n-1), \gamma(n), 1)$

endfor

end

This algorithm constructs and applies $n^2 - n$ Givens rotations. A similar procedure **backchase** to that illustrated above can be used to produces orthogonal matrices

$\bar{U}, \bar{V} \in \mathcal{R}^{n \times n}$ such that

$$\bar{B} = \bar{U}^T B \bar{V}, \quad \bar{V}^T z = \rho e_n, \quad \rho = \|z\| \tag{3.2}$$

where $\bar{B}$ is upper bidiagonal.

For the sake of brevity, we do not present **backchase**. It is computed by reversing the two vectors $\gamma(1{:}n)$ and $\phi(1{:}n-1)$, performing **forchase** and reversing the vectors back. The algorithm would just have the loop in **forchase** go backward instead of forward.

3.2.2   A Chasing Routine for a Lower Triangular Matrix

We now describe a simple chasing routine for a lower triangular matrix. This routine, for given a vector z and a lower triangular matrix C, finds orthogonal matrices $\bar{U}$ and $\bar{V}$ such that

$$\bar{C} = \bar{U}^T C \bar{V}, \quad \bar{V}^T z = \rho e_1, \quad \rho = \|z\| \tag{3.3}$$

where $\bar{C}$ is lower triangular.

Consider the 4×4 case in Fig. 3.2. We state the procedure **lchase** formally below.

ALGORITHM 3.5. Given a lower triangular matrix C and the updating vector z, this procedure performs a chasing operation on C, and produces a lower triangular matrix $\bar{C}$ that satisfies (3.3).

procedure lchase(c, z, n)

    for $i \leftarrow n - 1, \ldots, 1$

        formrot $(z(i), z(i+1), cn, sn)$;

        $e \leftarrow -sn * c(i, i)$; $c(i, i) \leftarrow cn * c(i, i)$;

        applyrot$(c(i+1{:}n, i), c(i+1{:}n, i+1), cn, sn, n-i)$;

$$\begin{array}{c} \downarrow \; \downarrow \\ \begin{pmatrix} z & z & z & \widehat{z} \\ c & & & \\ c & c & & \\ c & c & c & \\ c & c & c & c \end{pmatrix} \begin{matrix} \rightarrow \\ \rightarrow \end{matrix} \begin{pmatrix} z & z & z & 0 \\ c & & & \\ c & c & & \\ c & c & c & \widehat{c} \\ c & c & c & c \end{pmatrix} \end{array} \quad \begin{array}{c} \downarrow \quad \downarrow \\ \begin{pmatrix} z & z & \widehat{z} & 0 \\ c & & & \\ c & c & & \\ c & c & c & \\ c & c & c & c \end{pmatrix} \begin{matrix} \rightarrow \\ \rightarrow \end{matrix} \begin{pmatrix} z & z & 0 & 0 \\ c & & & \\ c & c & \widehat{c} & \\ c & c & c & \\ c & c & c & c \end{pmatrix} \end{array}$$

$$\begin{array}{c} \downarrow \; \downarrow \\ \begin{pmatrix} z & \widehat{z} & 0 & 0 \\ c & & & \\ c & c & & \\ c & c & c & \\ c & c & c & c \end{pmatrix} \rightarrow \begin{pmatrix} z & 0 & 0 & 0 \\ c & \widehat{c} & & \\ c & c & & \\ c & c & c & \\ c & c & c & c \end{pmatrix} \begin{pmatrix} z & 0 & 0 & 0 \\ c & & & \\ c & c & & \\ c & c & c & \\ c & c & c & c \end{pmatrix} \end{array}$$

Fig. 3.2. Chasing Steps for a Lower Triangular Matrix

formrot $(c(i+1, i+1), e, cn, sn)$;

applyrot$(c(i+1, 1 : i), c(i, 1 : i), cn, sn, i)$;

  endfor

end

A similar chasing procedure can be specified for an upper triangular matrix when modifying the URVD. Stewart [102] points out that if the matrix C is from a rank revealing decomposition with k large rows and $n - k$ small rows, this algorithm can yield $k + 1$ large rows, thus the rank revealing nature of C may be lost.

3.3 qd Procedure

It is easy to find an orthogonal matrix Q as a product of Givens rotations such that

$$\widehat{B} = QB$$

where $\widehat{B}$ is lower bidiagonal and B is upper bidiagonal. This is the same as one unshifted Fernando-Parlett **qd** step [40], hence the name. Fig. 3.3 illustrates the reduction steps for a 4×4 case.

$$
\begin{matrix} \rightarrow \\ \rightarrow \end{matrix}
\begin{pmatrix} b & & & \\ \widehat{b} & b & & \\ & b & b & \\ & & b & b \end{pmatrix} \rightarrow
\begin{pmatrix} b & b & & \\ & b & & \\ & \widehat{b} & b & \\ & & b & b \end{pmatrix} \begin{matrix} \rightarrow \\ \rightarrow \end{matrix}
\begin{pmatrix} b & b & & \\ & b & b & \\ & & b & \\ & & \widehat{b} & b \end{pmatrix}
\begin{pmatrix} b & b & & \\ & b & b & \\ & & b & b \\ & & & b \end{pmatrix}
$$

Fig. 3.3. One Step of qd Procedure

ALGORITHM 3.6. This procedure produces the orthogonal factorization of a lower bidiagonal matrix. $\gamma(1\colon n)$ is the diagonal on input and output. $\phi(1\colon n-1)$ is the subdiagonal on input and the superdiagonal on output.

procedure qd (γ, ϕ, n)

    for $i \leftarrow 1, \ldots, n-1$

        formrot$(\gamma(i), \phi(i), cn, sn)$

        $\phi(i) \leftarrow sn * \gamma(i+1)$

        $\gamma(i+1) \leftarrow cn * \gamma(i+1)$

    endfor

end

3.4   The LINPACK Downdating Procedure

The following downdating procedure due to Saunders [46] is considered the most accurate downdating procedure that does not require information from the first row of U in (1.1) [20] (if we have that first row, we obtain a procedure that is backward stable in the strong sense). It is the procedure that is implemented in the LINPACK [37].

ALGORITHM 3.7 (THE LINPACK DOWNDATING PROCEDURE). Given $M \in \mathcal{R}^{n \times n}$ and $z \in \mathcal{R}^n$, this algorithm computes the downdated matrix $\bar{M}$, that is, $\bar{M}^T \bar{M} = M^T M - zz^T$.

Step 1. Solve

$$M^T a = z \qquad (3.4)$$

If $\|a\| \geq 1$ declare $M^T M - zz^T$ indefinite and stop. Otherwise go to Step 2.

Step 2. Compute $\alpha = \sqrt{1 - \|a\|^2}$ and $Q = Q_1 \cdots Q_n \in \mathcal{R}^{(n+1) \times (n+1)}$ be a product of Givens rotations, $Q_i = J(1, i+1, \theta_i)$, $i = 1, \ldots, n$ such that

$$Q \begin{pmatrix} \alpha \\ a \end{pmatrix} = e_1.$$

Step 3. Compute

$$Q \begin{pmatrix} 0 \\ M \end{pmatrix} = \begin{pmatrix} z^T \\ \bar{M} \end{pmatrix}.$$

We note that if M is upper or lower triangular, it is simple to choose Q as a product of Givens rotations $Q_1, Q_2, \ldots, Q_n$ so that $\bar{M}$ remains upper or lower triangular. Pan [85] shows that for M upper triangular, this method can be sped up by combining the forward substitution phase with the application of the Givens rotations.

3.5 2×2 Updating/Downdating Procedure

The following algorithm computes orthogonal matrices $G_1 = J(1,2,\phi_1)$, $G_2 = J(1,3,\phi_2) \in \mathcal{R}^{3\times3}$ for some ϕ_i, $i = 1,2$, such that

$$\begin{pmatrix} 0 & 0 \\ \tilde{a} & \tilde{b} \\ 0 & \tilde{c} \end{pmatrix} = G_2^T G_1^T \begin{pmatrix} \xi & \rho \\ a & b \\ 0 & c \end{pmatrix} \tag{3.5}$$

ALGORITHM 3.8 (2×2 UPDATING). Given scalars a, b, c, ξ, ρ, this algorithm computes scalars $\tilde{a}, \tilde{b}, \tilde{c}$ of an updated matrix defined in (3.5).

function $[\tilde{a}, \tilde{b}, \tilde{c}] = $ **up22** (a,b,c,ξ,ρ)

   $\tilde{a} = $ **cabs**(a,ξ); $\bar{a} = a/\tilde{a}$; $\bar{\xi} = \xi/\tilde{a}$

   $\tilde{b} = \bar{a}b - \bar{\xi}\rho$; $\tilde{c} = \bar{a}\rho - \bar{\xi}b$

end

Similarly we give an algorithm, which is based on Algorithm 3.7 for downdating a 2×2 matrix, that is, computing orthogonal matrices $Q_1 = J(1,3,\theta_1)$, $Q_2 = J(1,2,\theta_2) \in \mathcal{R}^{3\times3}$ such that

$$\begin{pmatrix} 1 & \xi & \rho \\ 0 & \tilde{a} & \tilde{b} \\ 0 & 0 & \tilde{c} \end{pmatrix} = Q_2^T Q_1^T \begin{pmatrix} \alpha & 0 & 0 \\ \beta & a & b \\ \gamma & 0 & c \end{pmatrix}, \tag{3.6}$$

where we solve

$$\begin{pmatrix} a & b \\ 0 & c \end{pmatrix}^T \begin{pmatrix} \beta \\ \gamma \end{pmatrix} = \begin{pmatrix} \xi \\ \rho \end{pmatrix}, \quad \alpha = \sqrt{1 - \beta^2 - \gamma^2}, \tag{3.7}$$

where $J(i, k, \theta)$ is a Givens rotation in the (i, k) coordinate plane for some θ.

ALGORITHM 3.9 (2×2 DOWNDATING). Given scalars a, b, c, β, and γ, this algorithm computes entries of downdated matrix defined in (3.6), namely, $\widetilde{a}$, $\widetilde{b}$, and $\widetilde{c}$.

function $[\widetilde{a}, \widetilde{b}, \widetilde{c}] = $ **down22** (a, b, c, β, γ)

$\quad \widetilde{\alpha} = \sqrt{1 + \beta}\sqrt{1 - \beta}$

$\quad \alpha = \sqrt{\widetilde{\alpha} + \gamma}\sqrt{\widetilde{\alpha} - \gamma}$

$\quad \widetilde{\rho} = \gamma c / \widetilde{\alpha}$

$\quad \widetilde{c} = \alpha c / \widetilde{\alpha}; \quad \widetilde{a} = \widetilde{\alpha} a; \quad \widetilde{b} = \widetilde{\alpha} b - \widetilde{\rho}\beta$

end

Chapter 4

Modifying the ULV Decomposition

4.1 Introduction

We give methods for modifying the ULVD. Rewrite the ULVD of $A \in \mathcal{R}^{m \times n}$ of (1.1) as

$$A = U \begin{pmatrix} C \\ 0 \end{pmatrix} V^T \tag{4.1}$$

where $U \in \mathcal{R}^{m \times m}$ and $V \in \mathcal{R}^{n \times n}$ are orthogonal, and

$$C = \begin{pmatrix} \overset{k}{L} & \overset{p-k}{0} & \overset{n-p}{0} \\ F & G & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{matrix} k \\ p-k \\ n-p \end{matrix} \tag{4.2}$$

so that our algorithm of ULVD separates out columns that are exactly zero. Here C takes the position of M in (1.1).

We also rewrite z of (1.9) as

$$z = V^T r = \begin{pmatrix} x \\ y \\ y_0 \end{pmatrix} \begin{matrix} k \\ p-k \\ n-p \end{matrix} \tag{4.3}$$

As in the updating routine of Stewart [102], the matrices $\bar{C}$ and $\bar{V}$ can be produced using $\mathcal{O}(n)$ Givens rotations, thus updating the factorization in $\mathcal{O}(n^2)$ operations. Our approach to downdating the ULVD (4.2) uses ideas from chasing algorithms [102] and from the downdating algorithm due to Saunders [46, 85].

The following are the main results of this chapter:

1. A blockwise procedure for downdating the ULVD that yields

$$\bar{C} = \begin{pmatrix} \bar{L} & 0 & 0 \\ \bar{F} & \bar{G} & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{4.4}$$

where

$$\|(\bar{F} \ \bar{G})\| \le \|(F \ G)\|, \quad \|(\bar{F} \ \bar{G})\|_F \le \|(F \ G)\|_F$$

and $\bar{L}$ and $\bar{G}$ are lower triangular, and the blocks are conformal with (4.2).

2. A downdating algorithm that works whenever $L^T L - xx^T$ is positive definite, the same as if we were downdating only L. Our technique maps back onto the original matrix A in a more satisfactory manner than the technique given by Park and Eldén [87].

3. An error analysis of this procedure showing that the singular subspaces of the updated matrix are as good as can be expected. We also give some new perturbation results showing that the condition of the downdate is related only to the $\bar{L}$ block in (4.4). Thus tracking the ULVD is a very stable method for tracking subspaces.

Our ULVD downdating algorithm is proposed in detail in Section 4.2. Section 4.3 gives an error analysis of the algorithm. The accuracy of the computed subspaces for

large and small singular values is assessed. In Section 4.4, we give numerical tests of our algorithm on recursive total least squares problems.

4.2 A Procedure for Downdating ULV Decomposition

4.2.1 Description of the Algorithm

We introduce the following algorithm for downdating the ULVD. Our procedure produces a downdated matrix $\bar{C}$ if for L and x defined in (4.2) and (4.3), $L^T L - xx^T$ is positive definite.

ALGORITHM 4.1 (PROCEDURE FOR ULVD DOWNDATING). Given a lower triangular matrix C of the form (4.2) and a vector z of the form (4.3), this algorithm finds a lower triangular matrix $\bar{C}$ of the form (4.4) and orthogonal matrices $\bar{U}$ and $\bar{V}$ satisfying (1.12). Initially, $\bar{V} = V$. The components in y_0 are ignored (will be justified by Proposition 4.2). Throughout the description of this algorithm $L^{(i)}$ and $G^{(i)}$ denote lower triangular matrices, $f_1^{(i)} = [F^{(i)}]^T e_1$, and $g_{11}^{(i)} = e_1^T [G^{(i)}] e_1$.

Step 1. Compute orthogonal matrices $\bar{U}_1, \bar{V}_1 \in \mathcal{R}^{(p-k) \times (p-k)}$ such that

$$G^{(1)} = \bar{U}_1^T G \bar{V}_1, \quad \bar{V}_1^T y = \rho e_1^{(p-k)}, \quad \rho = \|y\|.$$

Also, compute

$$F^{(1)} = \bar{U}_1^T F.$$

Update $\bar{U} \leftarrow \text{diag}(I_{k+1}, \bar{U}_1, I_{n-p}); \bar{V} \leftarrow \bar{V} \text{diag}(I_k, \bar{V}_1, I_{n-p}).$

Step 2. Find an orthogonal matrix $\bar{U}_2 \in \mathcal{R}^{(k+1)\times(k+1)}$ such that

$$\begin{pmatrix} L^{(1)} & h \\ 0 & g_{11}^{(2)} \end{pmatrix} = \bar{U}_2^T \begin{pmatrix} L & 0 \\ \{f_1^{(1)}\}^T & g_{11}^{(1)} \end{pmatrix}.$$

Define $F^{(2)} = (I - e_1 e_1^T)F^{(1)}$, that is, as $F^{(1)}$ with its first row set to zero. Update $\bar{U} \leftarrow \bar{U} \operatorname{diag}(1, \bar{U}_2, I_{n-k-1})$.

Step 3. Use Algorithm 3.7 to find a vector $a \in \mathcal{R}^k$, scalar α, and orthogonal matrix $\bar{U}_3 \in \mathcal{R}^{(k+1)\times(k+1)}$ such that

$$[L^{(1)}]^T a = x, \quad \alpha = \sqrt{1 - \|a\|^2}, \quad \bar{U}_3^T \begin{pmatrix} \alpha \\ a \end{pmatrix} = e_1. \tag{4.5}$$

Then compute

$$\begin{pmatrix} x^T \\ L^{(2)} \end{pmatrix} = \bar{U}_3^T \begin{pmatrix} 0 \\ L^{(1)} \end{pmatrix}.$$

Step 4. Compute

$$\tilde{\rho} = \alpha^{-1}(\rho - h^T a)$$

if $\tilde{\rho} \le g_{11}^{(2)}$ then

$$\begin{pmatrix} \rho \\ \tilde{h} \end{pmatrix} = \bar{U}_3^T \begin{pmatrix} \tilde{\rho} \\ h \end{pmatrix}, \quad g_{11}^{(3)} = \sqrt{[g_{11}^{(2)}]^2 - \tilde{\rho}^2}$$

else $g_{11}^{(2)} < \tilde{\rho}$

$$\begin{pmatrix} \rho - \delta\rho \\ \tilde{h} \end{pmatrix} = \bar{U}_3^T \begin{pmatrix} g_{11}^{(2)} \\ h \end{pmatrix}, \quad g_{11}^{(3)} = 0 \qquad (4.6)$$

where $\delta\rho = \rho - (\alpha g_{11}^{(2)} + a^T h)$.

endif

Define $\bar{U}_4 = J(1, k+1, \theta)$ as the Givens rotation with $cn = \cos(\theta)$, $sn = \sin(\theta)$ such that

$$cn = \begin{cases} g_{11}^{(3)}/g_{11}^{(2)} & \text{if } g_{11}^{(2)} \neq 0 \\ 0 & \text{if } g_{11}^{(2)} = 0. \end{cases}$$

Update $\bar{U} \leftarrow \bar{U}\bar{U}_4 \operatorname{diag}(\bar{U}_3, I_{n-k})$.

Step 5. Find an orthogonal matrix $\bar{V}_2 \in \mathcal{R}^{(k+1)\times(k+1)}$ such that

$$(L^{(3)} \quad 0) = (L^{(2)} \quad \tilde{h}e_1^T)\bar{V}_2$$
$$(F^{(3)} \quad G^{(4)}) = (F^{(2)} \quad G^{(3)})\operatorname{diag}(\bar{V}_2, I_{n-k-1})$$

Update $\bar{V} \leftarrow \bar{V}\operatorname{diag}(\bar{V}_2, I_{n-k-1})$. If $g_{11}^{(4)} \neq 0$,

$$\bar{C} = \begin{pmatrix} \bar{L} & 0 & 0 \\ \bar{F} & \bar{G} & 0 \\ 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} L^{(3)} & 0 & 0 \\ F^{(3)} & G^{(4)} & 0 \\ 0 & 0 & 0 \end{pmatrix},$$

and go to Step 7; otherwise, go to Step 6.

Step 6. If $g_{11}^{(4)} = 0$ then $f_1^{(3)} = 0$ also since it was formed from $g_{11}^{(3)}$ using $\bar{V}_2$. Thus

$$
\begin{array}{c}
\begin{array}{cc} k & p-k \end{array} \\
\begin{array}{c} k \\ p-k \end{array}
\begin{pmatrix} L^{(3)} & 0 \\ F^{(3)} & G^{(4)} \end{pmatrix}
\end{array}
=
\begin{array}{c}
\begin{array}{cc} k & p-k \end{array} \\
\begin{pmatrix} \bar{L} & 0 \\ 0 & 0 \\ \bar{F} & \widetilde{G}^{(4)} \end{pmatrix}
\begin{array}{c} k \\ 1 \\ p-k-1 \end{array}
\end{array}
$$

where $\widetilde{G}^{(4)}$ is a lower Hessenberg matrix. We then find an orthogonal matrix $\bar{V}_3 \in \mathcal{R}^{(p-k)\times(p-k)}$ and an orthogonal permutation matrix $\bar{U}_5 \in \mathcal{R}^{(p-k)\times(p-k)}$ such that

$$
\bar{U}_5^T G^{(4)} \bar{V}_3 =
\begin{pmatrix} \bar{G} & 0 \\ 0 & 0 \end{pmatrix}
\begin{array}{c} p-k-1 \\ 1 \end{array}
$$

Update $\bar{U} \leftarrow \bar{U} \operatorname{diag}(I_{k+1}, \bar{U}_5, I_{n-p})$; $\bar{V} \leftarrow \bar{V} \operatorname{diag}(I_k, \bar{V}_3, I_{n-p})$. Thus $\bar{C}$ has the form

$$
\bar{C} =
\begin{array}{c}
\begin{array}{ccc} k & p-k-1 & n-p+1 \end{array} \\
\begin{pmatrix} \bar{L} & 0 & 0 \\ \bar{F} & \bar{G} & 0 \\ 0 & 0 & 0 \end{pmatrix}
\begin{array}{c} k \\ p-k-1 \\ n-p+1 \end{array}
\end{array}
$$

Step 7. Perform a ULVD of $\bar{L}$ to determine its numerical rank. If we have determined the rank of L correctly, the rank of $\bar{L}$ should be k or $k-1$. Make appropriate adjustments to $\bar{F}$ and $\bar{G}$.

We give an expression for $\bar{U}$ in the statement of the algorithm, but our analysis assumes that the left orthogonal matrix U is not saved. Although it is not computed by

the algorithm, for the discussion that follows we need to define the vector $\bar{z}$ by

$$\bar{z} = \bar{V}^T z = \begin{pmatrix} \bar{x} \\ \bar{y} \\ y_0 \end{pmatrix} \begin{matrix} k \\ p-k \\ n-p \end{matrix} \quad . \tag{4.7}$$

Algorithm 4.1 requires $11k^2 + 6(p-k)^2 + 12k(p-k) + \mathcal{O}(p)$ flops for Steps 1-5 and $4k^2 + \mathcal{O}(k)$ flops for Step 6. When V is modified, additional $6n(p-k) + 6nk$ flops will be required. The deflation step requires about $\nu k^2 + 16nk$ flops, where ν is the constant depending upon the condition estimators used [61].

Fig. 4.1–4.2 illustrate the action of our algorithm on a 6 × 6 matrix with $k = 3$. Here l, g, and f denote components of L, F and G, and x, y, and h are components of those vectors. Rightarrows and downarrows denote premultiplication and postmultiplication by orthogonal transformations from the left and from the right, respectively. Here ρ, h and $\tilde{h}$ have meanings consistent with those in the statement of Algorithm 4.1.

A set of $\overset{*}{\rightarrow}$ denotes the applications of $\bar{U}_3$ and $\bar{U}_4$ in Steps 3 and 4 when downdating L and $g_{11}^{(2)}$. Step 6 is illustrated in Fig. 4.2. Note that for Steps 3–5, the components x and y have a different interpretation from Steps 1 and 2 above, they are now the values "downdated." To illustrate their action properly, $\bar{U}_3$ and $\bar{U}_4$ are given in reverse order rather than the order that they are actually applied.

REMARK 4.1. This algorithm works if

$$\begin{matrix} p-k & n-p \end{matrix}$$
$$\begin{pmatrix} G & 0 \\ 0 & 0 \end{pmatrix} \begin{matrix} p-k \\ n-p \end{matrix}$$

$$
\begin{pmatrix}
x & x & x & y & y & \widehat{y} \\
l & & & & & \\
l & l & & & & \\
l & l & l & & & \\
f & f & f & g & & \\
f & f & f & g & g & \\
f & f & f & g & g & g
\end{pmatrix}
\begin{matrix} \to \\ \to \end{matrix}
\begin{pmatrix}
x & x & x & y & y & 0 \\
l & & & & & \\
l & l & & & & \\
l & l & l & & & \\
f & f & f & g & & \\
f & f & f & g & g & \widehat{g} \\
f & f & f & g & g & g
\end{pmatrix}
\begin{pmatrix}
x & x & x & y & \widehat{y} & 0 \\
l & & & & & \\
l & l & & & & \\
l & l & l & & & \\
f & f & f & g & & \\
f & f & f & g & g & \\
f & f & f & g & g & g
\end{pmatrix}
$$

$$
\begin{matrix} \to \\ \to \end{matrix}
\begin{pmatrix}
x & x & x & \rho & 0 & 0 \\
l & & & & & \\
l & l & & & & \\
l & l & l & & & \\
f & f & f & g & \widehat{g} & \\
f & f & f & g & g & \\
f & f & f & g & g & g
\end{pmatrix}
\begin{matrix} \to \\ \to \end{matrix}
\begin{pmatrix}
x & x & x & \rho & 0 & 0 \\
l & & & & & \\
l & l & & & & \\
l & l & l & & & \\
f & f & \widehat{f} & g & & \\
f & f & f & g & g & \\
f & f & f & g & g & g
\end{pmatrix}
\to
\begin{pmatrix}
x & x & x & \rho & 0 & 0 \\
l & & & & & \\
l & l & & & & \\
l & l & l & h & & \\
f & \widehat{f} & 0 & g & & \\
f & f & f & g & g & \\
f & f & f & g & g & g
\end{pmatrix}
$$

$$
\to
\begin{pmatrix}
x & x & x & \rho & 0 & 0 \\
l & & & & & \\
l & l & & h & & \\
l & l & l & h & & \\
\widehat{f} & 0 & 0 & g & & \\
f & f & f & g & g & \\
f & f & f & g & g & g
\end{pmatrix}
\begin{pmatrix}
x & x & x & \rho & 0 & 0 \\
l & & & h & & \\
l & l & & h & & \\
l & l & l & h & & \\
0 & 0 & 0 & g & & \\
f & f & f & g & g & \\
f & f & f & g & g & g
\end{pmatrix}
$$

(a) Steps 1 and 2

$$
\begin{matrix} \to \\ \to \\ \to \\ \to \\ \to \end{matrix}
\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 \\
l & & & h & & \\
l & l & & h & & \\
l & l & l & h & & \\
0 & 0 & 0 & g & & \\
f & f & f & g & g & \\
f & f & f & g & g & g
\end{pmatrix}
\begin{pmatrix}
x & x & x & y & y & y \\
l & & & \widehat{h} & & \\
l & l & & \widetilde{h} & & \\
l & l & l & \widetilde{h} & & \\
0 & 0 & 0 & g & & \\
f & f & f & g & g & \\
f & f & f & g & g & g
\end{pmatrix}
\begin{pmatrix}
x & x & x & y & y & y \\
l & & & & & \\
l & l & & \widehat{h} & & \\
l & l & l & \widetilde{h} & & \\
f & 0 & 0 & g & & \\
f & f & f & g & g & \\
f & f & f & g & g & g
\end{pmatrix}
$$

$$
\begin{pmatrix}
x & x & x & y & y & y \\
l & & & & & \\
l & l & & & & \\
l & l & l & \widehat{h} & & \\
f & f & 0 & g & & \\
f & f & f & g & g & \\
f & f & f & g & g & g
\end{pmatrix}
\begin{pmatrix}
\bar{x} & \bar{x} & \bar{x} & \bar{y} & \bar{y} & \bar{y} \\
l & & & & & \\
l & l & & & & \\
l & l & l & & & \\
f & f & f & g & & \\
f & f & f & g & g & \\
f & f & f & g & g & g
\end{pmatrix}
$$

(b) Steps 3–5

Fig. 4.1.   Reduction Steps for Downdating the ULVD

$$
\begin{matrix}
 & & & \downarrow & \downarrow & \\
\begin{pmatrix}
x & x & x & y & y & y \\
l & & & & & \\
l & l & & & & \\
l & l & l & & & \\
0 & 0 & 0 & 0 & & \\
f & f & f & g & \widehat{g} & \\
f & f & f & g & g & g
\end{pmatrix}
&
\begin{matrix}
 & & & \downarrow & \downarrow & \\
\begin{pmatrix}
x & x & x & y & y & y \\
l & & & & & \\
l & l & & & & \\
l & l & l & & & \\
0 & 0 & 0 & 0 & & \\
f & f & f & g & & \\
f & f & f & g & g & \widehat{g}
\end{pmatrix}
\end{matrix}
&
\begin{matrix} \rightarrow \\ \rightarrow \end{matrix}
&
\begin{pmatrix}
x & x & x & y & y & y \\
l & & & & & \\
l & l & & & & \\
l & l & l & & & \\
0 & 0 & 0 & 0 & & \\
f & f & f & g & & \\
f & f & f & g & g &
\end{pmatrix}
\end{matrix}
$$

$$
\begin{matrix}
\begin{matrix} \rightarrow \\ \rightarrow \end{matrix}
&
\begin{pmatrix}
x & x & x & y & y & y \\
l & & & & & \\
l & l & & & & \\
l & l & l & & & \\
f & f & f & g & & \\
0 & 0 & 0 & 0 & & \\
f & f & f & g & g &
\end{pmatrix}
&
\begin{pmatrix}
\bar{x} & \bar{x} & \bar{x} & \bar{y} & \bar{y} & \bar{y} \\
l & & & & & \\
l & l & & & & \\
l & l & l & & & \\
f & f & f & g & & \\
f & f & f & g & g & \\
0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
\end{matrix}
$$

Fig. 4.2. Reduction Steps When G Becomes Singular

is substituted for G and $(y^T\ y_0^T)^T$ is substituted for y. That is, it is not necessary to explicitly handle the zero block, it can be made part of G. That is the original formulation in [102]. However, if that is done, whenever $y_0 \neq 0$, any zero diagonal will be chased to the g_{11} position, all of $(y^T\ y_0^T)^T$ will be treated as "noise". If $\|G\| \gg \mu$, as is often true in practice, then $\|y\|$ is possibly significant whereas y_0 can only result from computing errors from computing (4.2) or multiplying $z = V^T r$. Our formulation neglects part of y only if the downdate of

$$
\begin{pmatrix}
L & 0 \\
F & G
\end{pmatrix}
$$

with $(x^T\ y^T)^T$ cannot be done. In updating, there is no similar benefit to separating out the zero block.

REMARK 4.2. We note that (4.6) in Step 4 is equivalent to computing

$$\begin{pmatrix} \rho \\ \tilde{h} \end{pmatrix} = \bar{U}_3^T \begin{pmatrix} g_{11}^{(2)} + (\delta\rho)\alpha \\ h + (\delta\rho)a \end{pmatrix}, \quad \delta\rho = \rho - (\alpha g_{11}^{(2)} + a^T h)$$

since $\bar{U}_3^T$ maps the additive noise vector $(\delta\rho)(\alpha \;\; a^T)^T$ into $(\delta\rho)e_1$. Only $\tilde{h}$ is of interest in the computation.

$\bar{C}$ satisfies (4.4) as is proven quickly in the following proposition.

PROPOSITION 4.1. *For the matrix $\bar{C}$ resulting from Algorithm 4.1 we have (4.4).*

Proof. This proof is a straightforward consequence of facts about orthogonal transformations. Every step except two and four either does not affect F and G or just multiplies them on the left or right by orthogonal transformations, thus for those steps we need only invoke orthogonal invariance.

For Step 2, we have that

$$F^{(2)} = F^{(1)} - e_1 e_1^T F^{(1)}, \quad \begin{pmatrix} h \\ g_{11}^{(2)} \end{pmatrix} = \bar{U}_2^T \begin{pmatrix} 0 \\ g_{11}^{(1)} \end{pmatrix}.$$

Thus $\|(F^{(2)} \;\; G^{(2)})\|_F \le \|(F^{(1)} \;\; G^{(1)})\|_F$ follows from the fact that $F^{(2)}$ and $G^{(2)}$ are no larger than $F^{(1)}$ and $G^{(1)}$ componentwise. For the Euclidean norm, we note that for any vector

$$v = \begin{pmatrix} v^{(1)} \\ v^{(2)} \end{pmatrix} \begin{matrix} k \\ p-k \end{matrix},$$

thus

$$\|(F^{(1)} \ G^{(1)})v\|^2 \ = \ \|(F^{(2)} \ G^{(2)})v\|^2 + (\{f_{(1)}\}^T v^{(1)})^2$$
$$+ (\{g_{11}^{(1)}\}^2 - \{g_{11}^{(2)}\}^2)(e_1^T v^{(2)})^2$$
$$\geq \ \|(F^{(2)} \ G^{(2)})v\|^2$$

for all $v \in \mathcal{R}^n$. A simple argument on the definition of the Euclidean norm yields the inequality $\|(F^{(1)} \ G^{(1)})\| \geq \|(F^{(2)} \ G^{(2)})\|$.

Step 4 clearly produces $|g_{11}^{(3)}| \leq |g_{11}^{(2)}|$. The same argument as above yields

$$\|(F^{(2)} \ G^{(3)})\| \leq \|(F^{(2)} \ G^{(2)})\|$$

and likewise for the Frobenius norm. Thus we have (4.4). □

In the absence of rounding error, we can show that the "additive noise" in Step 4 satisfies an important consistency property. We assume that C is orthogonally equivalent to $A + \delta A$ where δA consists of errors from previous orthogonal transformations.

PROPOSITION 4.2. *Assume that Algorithm 4.1 is performed in exact arithmetic, that U and V in (4.1) are exactly orthogonal. Let $\widetilde{U}$ and $\widetilde{V}$ be as in (1.13) and let $\bar{z} = \widetilde{V}^T r$ be as in (4.7). Assume that $\widetilde{U}e_1 = e_1$, and that $z = V^T r$ is computed exactly. Also assume that C satisfies (4.1) with backward error δA, that is*

$$A + \delta A = \begin{pmatrix} r^T + \delta r^T \\ \bar{A} + \delta \bar{A} \end{pmatrix} = U \begin{pmatrix} C \\ 0 \end{pmatrix} V^T. \tag{4.8}$$

Then $\bar{z}$ and $\bar{C}$ satisfy

$$A + \delta A_0 = \begin{pmatrix} r^T \\ \bar{A} + \delta\bar{A} \end{pmatrix} = \tilde{U} \begin{pmatrix} \bar{z}^T \\ \bar{C} \\ 0 \end{pmatrix} \tilde{V}^T. \qquad (4.9)$$

Thus $\|\delta A_0\| = \|\delta\bar{A}\| \leq \|\delta A\|$ and the same result holds for the Frobenius norm.

Proof. We have that

$$\bar{U}^T \begin{pmatrix} C \\ 0 \end{pmatrix} \mathrm{diag}(I_k, \bar{V}_1, I_{n-p}) = \begin{pmatrix} x + \delta x & (\rho - \delta\rho)e_1^T & 0 \\ L^{(2)} & \tilde{h}e_1^T & 0 \\ F^{(2)} & G^{(3)} & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

We also have that

$$\begin{pmatrix} \bar{z}^T \\ \bar{C} \end{pmatrix} \mathrm{diag}(I_k, \bar{V}_3^T, I_{n-p}) \mathrm{diag}(\bar{V}_2^T, I_{n-k-1}) = \begin{pmatrix} x & \rho e_1^T & y_0^T \\ L^{(2)} & \tilde{h}e_1^T & 0 \\ F^{(2)} & G^{(3)} & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

Let $\tilde{V}_1 = V \operatorname{diag}(I_k, \bar{V}_1, I_{n-p})$ then from the definition of $\bar{V}$ in Algorithm 4.1 it follows that

$$\tilde{U} \begin{pmatrix} \bar{z}^T \\ \bar{C} \\ 0 \end{pmatrix} \tilde{V}_1^T = \tilde{U} \begin{pmatrix} x & \rho e_1^T & y_0^T \\ L^{(2)} & \tilde{h}e_1^T & 0 \\ F^{(2)} & G^{(3)} & 0 \\ 0 & 0 & 0 \end{pmatrix} \operatorname{diag}(I_k, \bar{V}_1^T, I_{n-p}) V^T. \qquad (4.10)$$

However, we have

$$U \begin{pmatrix} C \\ 0 \end{pmatrix} V^T = \tilde{U} \begin{pmatrix} x + \delta x & (\rho - \delta\rho)e_1^T & 0 \\ L^{(2)} & \tilde{h}e_1^T & 0 \\ F^{(2)} & G^{(3)} & 0 \\ 0 & 0 & 0 \end{pmatrix} \operatorname{diag}(I_k, \bar{V}_1^T, I_{n-p}) V^T \quad (4.11)$$

$$= A + \delta A = \begin{pmatrix} r^T + \delta r^T \\ \bar{A} + \delta \bar{A} \end{pmatrix}. \qquad (4.12)$$

Comparing equations (4.10) and (4.11), using the fact that

$$V \operatorname{diag}(I_k, \bar{V}_1, I_{n-p}) \begin{pmatrix} x \\ \rho e_1 \\ y_0 \end{pmatrix} = r$$

and the assumption that $\tilde{U}e_1 = e_1$, we have (4.9). $\square$

Therefore, we have shown that the additive noise in Step 4 and the act of ignoring y_0 actually make the matrix $\bar{C}$ closer to being orthogonally equivalent to $\bar{A}$ than C is to

A. In Section 4.3, we show that the results of Propositions 4.1 and 4.2 make Algorithm 4.1 a robust algorithm for tracking the ULVD.

4.2.2 Relation to Park and Eldén's URV Procedure

A recent report by Park and Eldén [87] gives a method for downdating the URVD. For that algorithm, we are downdating an upper triangular matrix M of the form (1.4). The procedure is as follows.

1. Find an orthogonal matrix $\bar{U}_1$ and an upper triangular matrix $\bar{R}$ such that

$$
\begin{pmatrix} x^T \\ \bar{R} \end{pmatrix} = \bar{U}_1 \begin{pmatrix} 0 \\ R \end{pmatrix}
\tag{4.13}
$$

Determine $\bar{S}$ and $\tilde{y}$ such that

$$
\begin{pmatrix} y^T \\ \bar{S} \end{pmatrix} = \bar{U}_1 \begin{pmatrix} \tilde{y}^T \\ S \end{pmatrix}.
\tag{4.14}
$$

2. Find an orthogonal matrix $\bar{U}_2$ and an upper triangular matrix $\bar{T}$ such that

$$
\begin{pmatrix} \tilde{y}^T \\ \bar{T} \end{pmatrix} = \bar{U}_2 \begin{pmatrix} 0 \\ T \end{pmatrix}.
\tag{4.15}
$$

The downdated matrix $\bar{M}$ is

$$
\bar{M} = \begin{pmatrix} \bar{R} & \bar{S} \\ 0 & \bar{T} \end{pmatrix}
$$

as before.

Park and Eldén [87] recommend the use of hyperbolic rotations in (4.14). That can be avoided by a simple and well-known trick. Let $(\alpha_1 \ a_1^T)^T$ be the first column of $\bar{U}_1$ as determined in (4.13). Then we note that

$$y = (\widetilde{y} \ S^T) \begin{pmatrix} \alpha_1 \\ a_1 \end{pmatrix}$$

which implies that

$$\widetilde{y} = \alpha_1^{-1}(y - S^T a_1).$$

Once $\widetilde{y}$ is determined, we obtain $\bar{S}$ from

$$\begin{pmatrix} y^T \\ \bar{S} \end{pmatrix} = \bar{U}_1 \begin{pmatrix} \widetilde{y}^T \\ S \end{pmatrix}.$$

If $T^T T - \widetilde{y}\widetilde{y}^T$ is indefinite, Park and Eldén substitute for (4.15) the operation

$$\bar{U}_2^T \begin{pmatrix} 0 \\ T \end{pmatrix} \bar{V} = \begin{pmatrix} \rho e_n^T \\ \widehat{T} \end{pmatrix},$$

where $\bar{V}$ is an orthogonal matrix such that $\bar{V}^T \widetilde{y} = \rho e_n$ and so that $\widehat{T}$ is triangular. They then compute

$$\begin{pmatrix} \widetilde{y}^T \\ \bar{S} \end{pmatrix} = \bar{U}_1^T \begin{pmatrix} y^T \\ S \end{pmatrix} \bar{V}.$$

where $\bar{y} = \bar{V}^T y$. The downdated matrix $\bar{T} = \widehat{T}$ except for $\bar{t}_{qq}$, $q = n - k$. That entry satisfies

$$\bar{t}_{qq} = \begin{cases} 0 & \text{if } |\widehat{t}_{qq}| < |\rho| \\ \sqrt{\widehat{t}_{qq}^2 - \rho^2} & \text{otherwise} . \end{cases}$$

The first condition is identical to the case where $T^T T - \widetilde{y}\widetilde{y}^T$ is indefinite.

REMARK 4.3. Algorithm 4.1 is related to the above algorithm although the essential steps of them were developed independently. Steps 1 and 2 of Algorithm 4.1 reduce the ULVD downdating problem to that of downdating the $(k+1) \times (k+1)$ upper triangular matrix

$$\begin{array}{cc} k & 1 \end{array}$$
$$\begin{pmatrix} L^{(2)} & h \\ 0 & g_{11}^{(2)} \end{pmatrix} \begin{array}{c} k \\ 1 \end{array} . \tag{4.16}$$

The Park-Eldén algorithm applied to this matrix would perform Steps 3 and 4 of Algorithm 4.1 except that it would perform Step 4 according to

$$\bar{U}_3^T \begin{pmatrix} \widetilde{\rho} \\ h \end{pmatrix} = \begin{pmatrix} \rho \\ \widetilde{h} \end{pmatrix} \tag{4.17}$$

instead of according to (4.6). The consistency property in Proposition 4.2 does not hold if we use (4.17). That can be seen when (4.17) is placed into the proof. Equation (4.10) does not change, but we do have

$$U_3^T \begin{pmatrix} g_{11}^{(2)} \\ h \end{pmatrix} = \begin{pmatrix} \rho - [\delta\widetilde{\rho}]\alpha \\ \widetilde{h} - [\delta\widetilde{\rho}]a \end{pmatrix} \tag{4.18}$$

where $\delta\tilde{\rho} = \tilde{\rho} - g_{11}^{(2)} = \alpha^{-1}\delta\rho$ using terminology from Algorithm 4.1. We note that the additive noise is multiplied by a factor of $\alpha^{-1} \geq 1$, but that is not crucial. Equation (4.18) implies that equation (4.11) in the proof of Proposition 4.2 is replaced by

$$
U \begin{pmatrix} 0 \\ C \end{pmatrix} V^T = \tilde{U} \begin{pmatrix} x + \delta x & (\rho - [\delta\tilde{\rho}]\alpha)e_1^T & 0 \\ L^{(2)} & (\tilde{h} - [\delta\tilde{\rho}]a)e_1^T & 0 \\ F^{(2)} & G^{(3)} & 0 \\ 0 & 0 & 0 \end{pmatrix} \mathrm{diag}(I_k, \bar{V}_1^T, I_{n-p})V^T.
$$

Thus the consistency property of Proposition 4.2 does not hold.

The Park-Eldén URVD algorithm requires $5k^2 + 5(n-k)^2 + 8k(n-k) + \mathcal{O}(n)$ flops. This algorithm is fewer flops than is required to downdate the ULVD by Algorithm 4.1, but there is an important advantage to maintaining the ULVD instead of the URVD.

Both the URVD and the ULVD will tend to produce a V matrix such that

$$
V = (V_1 \ V_2), \quad V_1 \in \mathcal{R}^{n \times k}, \ V_2 \in \mathcal{R}^{n \times (n-k)}
$$

where $\mathrm{range}(V_1)$ and $\mathrm{range}(V_2)$ are approximations to the subspaces associated with the first k and last $n - k$ singular values of A respectively. However, as found by Fierro and Hansen [43], if F in (4.2) and S in (1.4) satisfy $\|F\| \approx \|S\|$ and if

$$
\inf_{\|x\|=1} \|Rx\| > \epsilon > \max_{\|y\|=1} \|Gy\|,
$$

then the ULVD yields a more accurate approximation of the desired subspaces of A.

4.3 Error Analysis

4.3.1 Error Bounds on Algorithm 4.1

We begin by showing that if $\widetilde{\rho} \leq g_{11}^{(2)}$ in Step 4 and $y_0 = 0$, then Algorithm 4.1 produces a matrix $\bar{C}$ such that

$$\bar{V}^T C^T C \bar{V} - (\bar{z} + \delta\bar{z})(\bar{z} + \delta\bar{z})^T = (\bar{C} + \delta\bar{C})^T (\bar{C} + \delta\bar{C}) \qquad (4.19)$$

for some orthogonal matrix $\bar{V}$. This is the so-called mixed stability criterion defined in Definition 2.7. If in Algorithm 4.1, $y_0 \neq 0$ or $\widetilde{\rho} > g_{11}^{(2)}$, then we are, in fact, producing $\bar{C}$ such that

$$\bar{V}^T C^T C \bar{V} - (\bar{z} + \delta\bar{z} + \delta z_0)(b + \delta\bar{z} + \delta z_0)^T = (\bar{C} + \delta\bar{C})^T (\bar{C} + \delta\bar{C})$$

for some orthogonal matrix $\bar{V}$. In the context of Algorithm 4.1 ,

$$\delta z_0 = -\operatorname{diag}(\bar{V}_2, I_{n-k-1}) \operatorname{diag}(I_k, \bar{V}_3, I_{n-p}) \begin{pmatrix} [\delta\rho]e_{k+1} \\ y_0 \end{pmatrix}, \qquad (4.20)$$

where $\delta\rho = \rho - (\alpha g_{11}^{(2)} + a^T h)$. Note that

$$\|\delta z_0\|^2 = |\delta\rho|^2 + \|y_0\|^2 = |\rho - (\alpha g_{11}^{(2)} + a^T h)|^2 + \|y_0\|^2.$$

We note that $\bar{V}_3 = I_{p-k}$ if Step 6 is not done. The effect of δz_0 is discussed in Appendix B of [13]. It is essentially the same as the effect of δb as bounded in Proposition 4.3. The analysis in this section will ignore that effect, it will assume that we are analyzing the problem (4.19).

We define the scaling matrix

$$\Delta = \text{diag}(I_k, \|G\|\,\|\bar{L}^{-1}\|I_{n-k}).$$ (4.21)

We expect the rounding errors from Algorithm 4.1 to be columnwise proportional to diagonals of Δ.

THEOREM 4.1. *Let Algorithm 4.1 be performed on a matrix C of the form (4.2) in floating point arithmetic with machine unit μ. Then there exist orthogonal matrices $\bar{U} \in \mathcal{R}^{(n+1)\times(n+1)}, \bar{V} \in \mathcal{R}^{n \times n}$ such that*

$$\begin{pmatrix} \bar{z} + \delta\bar{z} \\ \bar{C} + \delta\bar{C} \end{pmatrix} = \bar{U}^T \begin{pmatrix} 0 \\ C \end{pmatrix} \bar{V}$$

where

$$\delta_z = \|\Delta^{-1}\delta\bar{z}\|, \quad \delta_C = \|\delta\bar{C}\Delta^{-1}\|$$

$$\delta_z, \delta_C \leq \phi(p)\,\|C\|\,\mu + \mathcal{O}(\mu^2)$$

where Δ is defined in (4.21) and $\phi(p)$ is a modestly sized function of p.

Theorem 4.1 is proven in Appendix A of [13]. This theorem is somewhat similar to error bounds on orthogonal factorization of matrices with disproportionate rows [8, 12].

The following corollary gives the error bounds that we get if we use no structure of the problem (4.2) or the resulting matrix (4.4).

COROLLARY 4.1.

$$\left\| \begin{pmatrix} \delta\bar{z} \\ \delta\bar{C} \end{pmatrix} \right\| \leq \phi_C(p)\,\|C\|\,\mu + \mathcal{O}(\mu^2)$$ (4.22)

where $\phi_C(p)$ is a modestly growing function of p.

In the next section, we discuss the effect of these rounding errors on the singular vectors of the matrix $\bar{C}$.

4.3.2 Effect of Rounding Errors on Singular Vectors

A common reason for computing the ULVD is to separate subspaces associated with large and small singular values. For the ULVD, after downdating , there will be l large singular values where $l = k$ or $l = k - 1$. If $W = (W_1 \quad W_2)$, $W_1 \in \mathcal{R}^{n \times l}$, $W_2 \in \mathcal{R}^{n \times (n-l)}$ is the matrix of right singular vectors of $\bar{C}$, then the computed range(W_1) and range(W_2) should be as close as possible to the expected ranges. In this section, we show how reliable we can expect these subspaces to be. Our bounds are somewhat better if $l = k$, that is, if the downdate does not alter the rank.

We need to measure the effect of both δ_z and δ_C on the invariant subspaces associated with the l largest singular vectors and the $n - l$ smallest ones. We define $\widehat{C}$ and $\widetilde{C}$ as matrices such that

$$\widehat{C}^T \widehat{C} = \bar{V}^T C^T C \bar{V} - \bar{z}\bar{z}^T \tag{4.23}$$

$$\widetilde{C}^T \widetilde{C} = \bar{V}^T C^T C \bar{V} - (\bar{z} + \delta\bar{z})(\bar{z} + \delta\bar{z})^T. \tag{4.24}$$

Defining $\bar{C}$ as in (4.19) implies that

$$\widetilde{C} = \bar{C} + \delta\bar{C}. \tag{4.25}$$

It is also necessary to define ω by

$$\omega = \max\{1, \|[L^{(2)}]^{-T} x\|, \|\bar{L}^{-T} \bar{x}\|\}. \tag{4.26}$$

Note that the definition of ω involves only the L block and is related to the condition number given by Pan [86].

First, we need the following three technical lemmata.

LEMMA 4.1. *The vector $\tilde{h}$ and scalar ρ from Algorithm 4.1 satisfy*

$$\left\| \begin{pmatrix} \rho \\ \tilde{h} \end{pmatrix} \right\| \leq \|G\|. \tag{4.27}$$

Proof. We note that

$$\left\| \begin{pmatrix} \rho \\ \tilde{h} \end{pmatrix} \right\| = \left\| \begin{pmatrix} I_{k+1} & 0 \end{pmatrix} \bar{U}^T \begin{pmatrix} 0 \\ G \end{pmatrix} \bar{V}_1 \right\| \leq \|G\|,$$

which completes the proof. $\square$

LEMMA 4.2. *Let $\bar{z}$ be as in (4.7) resulting from Algorithm 4.1. Then*

$$\|\bar{y}\| \leq \|G\|(1 + \omega).$$

Proof. From Algorithm 4.1 and (4.20) we have that

$$\bar{z} = \begin{pmatrix} \bar{x} \\ \bar{y} \\ y_0 \end{pmatrix} = \operatorname{diag}(I_k, \bar{V}_3^T, I_{n-p}) \operatorname{diag}(\bar{V}_2^T, I_{n-k-1}) \operatorname{diag}(I_k, \bar{V}_1^T, I_{n-p}) \begin{pmatrix} x \\ y \\ y_0 \end{pmatrix}.$$

By the orthogonality of $\bar{V}_3$ we have

$$\|\bar{y}\| = |\bar{\rho}|,$$

where

$$\begin{pmatrix} \bar{x} \\ \bar{\rho} \end{pmatrix} = \bar{V}_2^T \begin{pmatrix} x \\ \rho \end{pmatrix}.$$

Now consider the least squares problem:

$$\min_{a \in \mathcal{R}^k} \left\| \begin{pmatrix} [L^{(2)}]^T \\ \tilde{h}^T \end{pmatrix} a - \begin{pmatrix} x \\ \rho \end{pmatrix} \right\|.$$

Since

$$\begin{pmatrix} \bar{L}^T \\ 0 \end{pmatrix} = \bar{V}_2^T \begin{pmatrix} [L^{(2)}]^T \\ \tilde{h}^T \end{pmatrix},$$

we have

$$|\bar{\rho}| = \min_{a \in \mathcal{R}^k} \left\| \begin{pmatrix} [L^{(2)}]^T \\ \tilde{h}^T \end{pmatrix} a - \begin{pmatrix} x \\ \rho \end{pmatrix} \right\|.$$

Let $\hat{a} = [L^{(2)}]^{-T} x$. Then from Lemma 4.1

$$\begin{aligned}
|\bar{\rho}| &\leq \left\| \begin{pmatrix} [L^{(2)}]^T \\ \tilde{h}^T \end{pmatrix} \hat{a} - \begin{pmatrix} x \\ \rho \end{pmatrix} \right\| = |\tilde{h}^T \hat{a} - \rho| \\
&\leq |\rho| + |\tilde{h}^T \hat{a}| \leq \|G\| + \|G\| \, \|[L^{(2)}]^{-T} x\| \\
&\leq \|G\|(1 + \omega),
\end{aligned}$$

which completes the proof. $\square$

It is necessary to bound the effect of Δ and $\bar{z}$ on the singular vectors of $\bar{C}$.

LEMMA 4.3. *Let* w_i, $i = 1, 2, \ldots, n$, *be the right singular vectors of* $\bar{C}$. *Let* Δ *be defined by (4.21), let* $\bar{z}$ *be as in Lemma 4.2, and let* ω *be given by (4.26). Then*

$$\|\Delta w_i\| \leq \|\bar{L}^{-1}\| \sqrt{\sigma_i^2 + \|G\|^2} \leq \|\bar{L}^{-1}\| \, (\sigma_i + \|G\|), \quad i = 1, 2, \ldots, p$$

$$|\bar{z}^T w_i| \leq \omega \, (\sigma_i + 2 \, \|G\|), \quad i = 1, 2, \ldots, p.$$

Proof. We note that for $i = 1, 2, \ldots, p$,

$$w_i = \begin{pmatrix} w_i^{(1)} \\ w_i^{(2)} \\ 0 \end{pmatrix} \begin{matrix} k \\ p-k \\ n-p \end{matrix}$$

It is obvious from the form of $\bar{C}$, that the last $n - p$ components of all its nonzero singular vectors will be zero. It should also be noted that $w_i = e_i$, $i = p+1, \ldots, n$ form a singular subspace for the last $n - p$ zero singular values. For $i = 1, 2, \ldots, p$, we have that

$$\|\Delta w_i\|^2 = \|w_i^{(1)}\|^2 + \|G\|^2 \|\bar{L}^{-1}\|^2 \|w_i^{(2)}\|^2.$$

Taking square roots establishes the first bound on $\|\Delta w_i\|$. Continuing we have

$$\begin{aligned} \|\Delta w_i\|^2 &= \|\bar{L}^{-1} \bar{L} w_i^{(1)}\|^2 + \|G\|^2 \|\bar{L}^{-1}\|^2 \\ &\leq \|\bar{L}^{-1}\|^2 (\|\bar{L} w_i^{(1)}\|^2 + \|G\|^2) \end{aligned}$$

$$\leq \quad \|\bar{L}^{-1}\|^2(\sigma_i^2 + \|G\|^2)$$

$$\leq \quad \|\bar{L}^{-1}\|^2(\sigma_i + \|G\|)^2.$$

Taking square roots again establishes the second bound on $\|\Delta w_i\|$.

Now

$$
\begin{aligned}
|\bar{z}^T w_i| &\leq |\bar{x}^T w_i^{(1)}| + |\bar{y}^T w_i^{(2)}| \\
&\leq |\bar{x}^T \bar{L}^{-1} \bar{L} w_i^{(1)}| + \|\bar{y}\| \\
&\leq \|\bar{L}^{-T} \bar{x}\| \, \sigma_i + \|\bar{y}\| \\
&\leq \omega \, (\sigma_i + \|G\|) + \|G\| \\
&\leq \omega \, (\sigma_i + 2 \, \|G\|),
\end{aligned}
$$

which completes the proof. $\square$

Now we give a perturbation bound on the effects of the backward errors $\|\delta \bar{z}\|$ and $\|\delta \bar{C}\|$.

LEMMA 4.4. *Assume the terminology of Lemma 4.3. Let w_i, $\hat{w}_i$, $\tilde{w}_i$, $i = 1, 2, \ldots, n$ be the right singular vectors of $\bar{C}$, $\hat{C}$, and $\tilde{C}$ in (4.19), (4.23), and (4.24) respectively. Let $\tilde{\sigma}_1 \geq \cdots \geq \tilde{\sigma}_n$ be the singular values of $\tilde{C}$, and let $\sigma_1 \geq \cdots \geq \sigma_n$ be the singular values of $\bar{C}$. If $\tilde{\sigma}_i > \tilde{\sigma}_j$ and $\sigma_i > \sigma_j$ we have*

$$|\tilde{w}_i^T \hat{w}_j| \leq \omega \, \|\delta \bar{z}\| \left(\frac{1}{\tilde{\sigma}_i - \tilde{\sigma}_j} + \frac{4\|G\|}{\tilde{\sigma}_i^2 - \tilde{\sigma}_j^2} \right) + \mathcal{O}(\|\delta \bar{z}\|^2) \qquad (4.28)$$

$$|w_i^T \tilde{w}_j| \leq \frac{\|\delta C\|}{\sigma_i - \sigma_j} + \mathcal{O}(\|\delta \bar{C}\|^2). \qquad (4.29)$$

Proof. Equation (4.29) is just a standard error bound based upon the perturbation of the eigenvectors of $\bar{C}^T \bar{C}$. From the Kato [69] expansion for eigenvectors we obtain

$$
\begin{aligned}
|\tilde{w}_i^T \hat{w}_j| &= \left| \frac{\tilde{w}_i^T (\bar{z}(\delta\bar{z})^T + (\delta\bar{z})\bar{z}^T + (\delta\bar{z})(\delta\bar{z})^T)\tilde{w}_j}{\tilde{\sigma}_i^2 - \tilde{\sigma}_j^2} \right| + \mathcal{O}(\|\delta\bar{z}\|^2) \\
&\leq \frac{\|\delta\bar{z}\| (|\bar{z}^T \tilde{w}_i| + |\bar{z}^T \tilde{w}_j|)}{\tilde{\sigma}_i^2 - \tilde{\sigma}_j^2} + \mathcal{O}(\|\delta\bar{z}\|^2).
\end{aligned}
$$

From Lemma 4.3 we have

$$
|\tilde{w}_i^T \hat{w}_j| \leq \omega \, \|\delta\bar{z}\| \frac{\sigma_i + \sigma_j + 4 \, \|G\|}{\tilde{\sigma}_i^2 - \tilde{\sigma}_j^2} + \mathcal{O}(\|\delta\bar{z}\|^2).
$$

An algebraic simplification leads to (4.28). □

Simply using the definition of the $\| \cdot \|_F$ norm leads to the following proposition. It tells us how good our subspaces will be if we only have a bound of the form (4.22).

PROPOSITION 4.3. *Assume the terminology of Lemma 4.3. Let $W = (W_1 \ W_2)$, $W_1 \in \mathcal{R}^{n \times l}$, $W_2 \in \mathcal{R}^{n \times (n-l)}$ be the matrix of right singular vectors of $\bar{C}$ and let $\widehat{W} = (\widehat{W}_1 \ \widehat{W}_2)$ be the corresponding matrix of right singular vectors of $\widehat{C}$. If $\sigma_l > \sigma_{l+1}$ then*

$$
\|W_1^T \widehat{W}_2\|_F \leq \sqrt{l(n-l)} \left[\frac{\omega \, \|\delta\bar{z}\| + \|\delta\bar{C}\|}{\sigma_l - \sigma_{l+1}} + \frac{4 \, \omega \, \|G\| \, \|\delta\bar{z}\|}{\sigma_l^2 - \sigma_{l+1}^2} \right] + \mathcal{O}\left(\left\| \begin{pmatrix} \delta\bar{z} \\ \delta\bar{C} \end{pmatrix} \right\|^2 \right).
$$

Proposition 4.3 applies even if the downdate changes the rank. The condition number ω in (4.26) depends solely upon the L-block, the matrix C does not have to be

well-conditioned or even full rank for the downdating problem to be well-conditioned as is required in previous analyses [86, 87]. If there is no rank change, that is if $l = k$, we can get an even better bound as shown below.

We define e and E according to

$$e = \Delta^{-1} \delta \bar{z} / \delta_z, \quad E = \delta \bar{C} \, \Delta^{-1} / \delta_C, \tag{4.30}$$

where δ_z and δ_C are the bounds from Theorem 4.1. Note that $\|e\| = \|E\| = 1$. First, a technical lemma characterizes the vector $\bar{z}$ that is downdated.

LEMMA 4.5. *Assume the results and terminology from Theorem 4.1. Let $\widetilde{w}_i, i = 1, 2, \ldots, n$ be the right singular vectors of $\widetilde{C}$ and let $\widehat{w}_i$, $i = 1, 2, \ldots, n$ be the right singular vectors of $\widehat{C}$. Let $\widetilde{\sigma}_1 \geq \widetilde{\sigma}_2 \geq \ldots \geq \widetilde{\sigma}_n$ be the singular values of $\widetilde{C}$. Then for all i and j such that $\sigma_i \neq \sigma_j$ we have*

$$|\widetilde{w}_i^T \widehat{w}_j| \leq \frac{\delta_z}{\widetilde{\sigma}_i^2 - \widetilde{\sigma}_j^2} (\|\Delta \widetilde{w}_j\| \, |\bar{z}^T \widetilde{w}_i| + \|\Delta \widetilde{w}_i\| \, |\bar{z}^T \widetilde{w}_j|) + \mathcal{O}(\delta_z^2). \tag{4.31}$$

Proof. From Kato[69], we have that

$$\widetilde{w}_i^T \widehat{w}_j = \delta_z \frac{\widetilde{w}_j^T \Delta (\bar{z} e^T + e \bar{z}^T) \Delta \widetilde{w}_i}{\widetilde{\sigma}_i^2 - \widetilde{\sigma}_j^2} + \mathcal{O}(\delta_z^2)$$

where e is defined in (4.30). Taking norms we obtain (4.31). □

LEMMA 4.6. *Assume the results and terminology from Theorem 4.1. Let w_i, $i = 1, 2, \ldots, n$ be the right singular vectors of $\bar{C}$ and let $\widetilde{w}_i$, $i = 1, 2, \ldots, n$ be the right singular vectors*

of $\widetilde{C} = \bar{C} + \delta\bar{C}$ as in (4.25). Let $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n$ be the singular values of $\bar{C}$. Then for all i and j such that $\sigma_i > \sigma_j$ we have

$$|w_i^T \widetilde{w}_j| \leq 2 \frac{\delta_C \, \|\bar{L}^{-1}\| \, (\sigma_j + \|G\|)}{\sigma_i - \sigma_j} + \mathcal{O}(\delta_C^2). \qquad (4.32)$$

Proof. Again using the Kato [69] expansion

$$w_i^T \widetilde{w}_j = \frac{w_i^T (\bar{C} \, \delta\bar{C} + \bar{C}^T \delta\bar{C}) w_j}{\sigma_i^2 - \sigma_j^2} + \mathcal{O}(\delta_C^2).$$

Using the definition of E in (4.30) we have

$$w_i^T \widetilde{w}_j = \delta_C \frac{w_i^T \Delta E \bar{C} w_j + w_j^T C^T E \Delta w_j}{\sigma_i^2 - \sigma_j^2} + \mathcal{O}(\delta_C^2).$$

Using norm inequalities we have

$$|w_i^T \widetilde{w}_j| \leq \delta_C \frac{\|\Delta w_i\| \, \|E\| \, \|\bar{C} w_j\| + \|C w_i\| \, \|E\| \, \|\Delta w_j\|}{\sigma_i^2 - \sigma_j^2} + \mathcal{O}(\delta_C^2). \qquad (4.33)$$

Using Lemma 4.3 and the fact that $\|\bar{C} w_i\| = \sigma_i$ yield

$$|w_i^T \widetilde{w}_j| \leq \delta_C \frac{\|\bar{L}^{-1}\| \left[(\sigma_i + \|G\|) \sigma_j + (\sigma_j + \|G\|) \sigma_i \right]}{\sigma_i^2 - \sigma_j^2} + \mathcal{O}(\delta_C^2).$$

Reorganizing leads to (4.32). □

Now we can bound the quantities in our lemma on the singular vectors. These bounds actually apply to the singular vectors of $\bar{C}$, but we will ignore second order effects and use these bounds for the singular vectors of $\widetilde{C}$.

Now we need a bound on the errors in the singular vectors of $\bar{C}$ from δ_z. To within rounding error, the same bounds as in Lemma 4.3 will hold for $\widetilde{w}_i$, $i = 1, 2, \ldots, n$. That is, $\|z^T \widetilde{w}_j\| \leq \omega \, (\widetilde{\sigma}_j + 2 \, \|G\|)$ and $\|\Delta \widetilde{w}_j\| \leq \|\bar{L}^{-1}\| \, (\widetilde{\sigma}_j + \|G\|)$.

LEMMA 4.7. *Assume the terminology of Lemma 4.3 and the results and terminology of Theorem 4.1. Assume that $\widetilde{\sigma}_i > \widetilde{\sigma}_j$. Then*

$$|\widetilde{w}_j^T \widehat{w}_i| \leq 2 \, \frac{\delta_z \omega \, \|\bar{L}^{-1}\|}{\widetilde{\sigma}_i - \widetilde{\sigma}_j} \left[\widetilde{\sigma}_j + 1.5 \, \|G\| + 2 \, \|G\|^2/(\widetilde{\sigma}_j + \widetilde{\sigma}_i) \right] + \mathcal{O}(\delta_z^2).$$

Proof. Combining Lemmata 4.3 and 4.5 yields

$$
\begin{aligned}
|\widetilde{w}_j^T \widehat{w}_i| &\leq \frac{\delta_z \omega \, \|\bar{L}^{-1}\|}{\widetilde{\sigma}_i^2 - \widetilde{\sigma}_j^2} \left[(\widetilde{\sigma}_i + 2 \, \|G\|)(\widetilde{\sigma}_j + \|G\|) \right. \\
&\quad \left. + (\widetilde{\sigma}_j + 2 \, \|G\|)(\widetilde{\sigma}_i + \|G\|) \right] + \mathcal{O}(\delta_z^2) \\
&= 2 \, \frac{\delta_z \omega \, \|\bar{L}^{-1}\|}{\widetilde{\sigma}_i^2 - \widetilde{\sigma}_j^2} (\widetilde{\sigma}_i \widetilde{\sigma}_j + 1.5 \, \|G\|(\widetilde{\sigma}_i + \widetilde{\sigma}_j) + 2 \, \|G\|^2) + \mathcal{O}(\delta_z^2) \\
&\leq 2 \, \frac{\delta_z \omega \, \|\bar{L}^{-1}\|}{\widetilde{\sigma}_i - \widetilde{\sigma}_j} \left[\widetilde{\sigma}_j + 1.5 \, \|G\| + 2 \, \|G\|^2/(\widetilde{\sigma}_i + \widetilde{\sigma}_j) \right] + \mathcal{O}(\delta_z^2),
\end{aligned}
$$

which completes the proof. $\square$

The proof of the following theorem is obvious from Lemma 4.7.

THEOREM 4.2. *Assume the hypothesis and terminology of Lemma 4.7. Let $\widetilde{W}$ be the matrix of singular vectors of $\widetilde{C}$ and let $\widehat{W}$ be the matrix of singular vectors of $\widehat{C}$. If*

$$\widetilde{W} = (\widetilde{W}_1 \quad \widetilde{W}_2), \quad \widehat{W} = (\widehat{W}_1 \quad \widehat{W}_2),$$

where $\widetilde{W}_1, \widehat{W}_1 \in \mathcal{R}^{n \times k}$, and $\widetilde{W}_2, \widehat{W}_2 \in \mathcal{R}^{n \times (n-k)}$ then

$$\|\widetilde{W}_1^T \widehat{W}_2\|_F \leq 2 \sqrt{k(n-k)} \frac{\delta_z \omega \|\bar{L}^{-1}\|}{\tilde{\sigma}_k - \tilde{\sigma}_{k+1}} \times$$

$$\left[\tilde{\sigma}_{k+1} + 1.5 \|G\| + 2 \|G\|^2 / (\tilde{\sigma}_k + \tilde{\sigma}_{k+1}) \right] + \mathcal{O}(\delta_z^2)$$

Analogously, we can bound the effect of δ_C.

THEOREM 4.3. *Assume the hypothesis and terminology of Lemma 4.6. Let $\widetilde{W}$ be the matrix of singular vectors of $\widetilde{C} = \bar{C} + \delta\bar{C}$ and let W be the matrix of singular vectors of $\bar{C}$. If*

$$W = (W_1 \quad W_2), \quad \widetilde{W} = (\widetilde{W}_1 \quad \widetilde{W}_2),$$

where $W_1, \widetilde{W}_1 \in \mathcal{R}^{n \times k}$, and $W_2, \widetilde{W}_2 \in \mathcal{R}^{n \times (n-k)}$ then

$$\|W_1^T \widetilde{W}_2\|_F \leq 2 \sqrt{k(n-k)} \frac{\delta_C \|\bar{L}^{-1}\| (\sigma_{k+1} + \|G\|)}{\sigma_k - \sigma_{k+1}} + \mathcal{O}(\delta_C^2).$$

The effect of δ_C is, in fact, somewhat less critical than that of δ_z as has been stated in other analyses of this problem [86, 87, 99]. We note that the error bounds in

Theorems 4.2 and 4.3 are relative gap bounds on the error in the subspaces similar to those in [11, 36].

If $\|\bar{L}^{-1}\| \, \|G\| \ll 1$, these bounds are a significant improvement over those in Proposition 4.3. This is one of the reasons for maintaining the property (4.4).

4.4   Numerical Examples

In this section, we present a few examples from numerical experiments. These tests were performed using MATLAB on a SPARCstation 5 in IEEE Standard double precision with machine precision $\approx 10^{-16}$. The algorithm employed the sliding window technique described in Section 2.7.3.

At each step of the sliding window method with the window size m_0, an $m_0 \times n$ data matrix is constructed from an $m \times n$ observation matrix A by adding a new row to the data matrix in the previous window and deleting the oldest row from it. In step j, the row $m_0 + j$ of the observation matrix is added and the row j is deleted, giving the data matrix A_j.

Computing the ULVD of initial data matrix is described in Section 2.5.2. Then Algorithm 4.1 takes the lower triangular matrix (middle part of the decomposition), the orthogonal matrix (right part) as initial input and the modifying vector r, and successively modifies these matrices at every window step. The vector $z = V^T r$ is computed at the beginning of each window step.

We tested our algorithms in the context of the total least squares (TLS) problems. See Section 2.7 for details. We used the TLS solutions from the Jacobi SVD as reference in checking the accuracy of the solution and rank estimates of our algorithms.

Fig. 4.3-4.5 show the rank estimates by our algorithm, which are identical with those of the Jacobi SVD algorithm. The horizontal axis represents the window steps and the vertical axis the numerical rank of the window matrix.

The distance between the subspaces is given in the next plot using the Definition 2.3. Let

$$A_j = Y_j \Sigma_j W_j^T, \quad W_j = (W_{j1} \quad W_{j2})$$

be the SVD of A_j computed by the one-sided Jacobi method at step j. Let

$$A_j = U_j C_j V_j^T, \quad V_j = (V_{j1} \quad V_{j2})$$

be the ULVD of A_j computed by Algorithm 4.1. Note that here we are discarding U_j. Finally, let

$$C_j = \bar{Y}_j \bar{\Sigma}_j \bar{W}_j^T, \quad \bar{W}_j = (\bar{W}_{j1} \quad \bar{W}_{j2})$$

by the SVD of C_j computed by the one-sided Jacobi method. Define $\widetilde{W}_j$ by

$$\widetilde{W}_j = (\widetilde{W}_{j1} \quad \widetilde{W}_{j2}) = V_j \bar{W}_j.$$

Define the angles between the subspaces

$$\sin \theta_1 = \|W_{j1}^T V_{j2}\|, \quad \sin(\theta_2) = \|\widetilde{W}_{j1}^T V_{j2}\|, \quad \sin(\theta_3) = \|W_{j1}^T \widetilde{W}_{j2}\|. \tag{4.34}$$

The angles θ_i, $i = 1, 2, 3$ represent, respectively, error between the true noise subspace from the Jacobi SVD and the approximate one from tracking the ULVD, the approximation error from the ULVD, and the subspace errors from ULVD subspace

tracking. The value $\sin(\theta_3)$ is one that is bounded by our error analysis. We plotted $\log_{10}(\sin(\theta_i)), i = 1, 2$ in solid and dotted lines, respectively, in the vertical axis of the second graph. It turned out that $\log_{10}(\sin(\theta_3))$ was almost indistinguishable from $\log_{10}(\sin(\theta_1))$, so we did not plot it. $\sin(\theta_2)$ is the approximation error discussed by Fierro and Hansen [43].

Finally, the TLS errors

$$\tau_j = \frac{\|x_j^{(SVD)} - x_j^{(ULVD)}\|}{\|x_j^{(SVD)}\|}$$

are given in logarithm in the last plot. Here, $x_j^{(SVD)}$ and $x_j^{(ULVD)}$, are the TLS solutions using the SVD and the ULVD, respectively.

For our condition estimators, we use the LINPACK condition estimator to approximate the left singular vector that corresponds to the smallest singular value, followed by inverse iteration using this approximate singular vector as the initial vector. The tests show that up to three steps of inverse iterations suffice the accuracy of the approximate smallest singular value required by the algorithm.

EXAMPLE 4.1. A, a 110-by-6 random matrix, b, a 110-by-1 random vector. Entries of A and b were chosen from a uniform distribution on the interval $(0, 1)$. 85 randomly chosen rows of $(A; b)$ were multiplied by $\gamma = 10^{-4}$ in order to vary the rank of the matrix, and $tol = 10^{-2}$. The window size p used was 12.

EXAMPLE 4.2. Same as Example 4.1 except that $\gamma = 10^{-8}$ and $tol = 10^{-6}$.

EXAMPLE 4.3. Same as Example 4.1 except that the matrix had an outlier of size 10^4 at $(18, 1)$ position.

The first plot shows that our algorithm estimated the numerical ranks correctly throughout the sliding window steps in spite of frequent rank changes. Although the errors in tiny singular values were relatively large, and the small singular values were almost always overestimated, they were close enough to correctly estimate the rank. Thus the rank estimates from our algorithm and those by the Jacobi SVD were identical. As expected, the errors in the TLS solution τ_j are almost exactly the same as the size of $\sin(\theta_1)$ in (4.34).

The second plot in each figure shows that the noise subspace error is very small giving accurate TLS solutions. The quantities in (4.34) are shown to be essentially identical indicating that the subspace errors from our algorithm are from the rounding errors, not approximation errors. We note that the Example 4.1 has greater error in the noise subspace than Example 4.2.

This is probably because Example 4.1 has only a small relative gap in the spectrum around $\gamma = 10^{-5}$ but a large relative gap around $\gamma = 10^{-8}$. However, for all of our examples, the approximation to the subspaces by the ULVD is very good. Since the error bounds on the distance between the noise subspaces depend on the $(k + 1)$-st singular value, the approximated singular subspace gets better as γ decreases as shown in Fig. 4.3-4.4.

The TLS solution errors behave very much the same as the noise subspace errors. From (2.18)-(2.19) it is not difficult to see that the TLS errors differ from the noise subspace errors only by a constant factor.

Moreover, the algorithm performs well even when G becomes singular (indicated by '*' in the first plot). We tested several other examples, and these results were typical.

Since our downdating procedures use the LINPACK downdating algorithm, it is not difficult to generate the cases where the algorithm breaks down when $\|a\| > 1$, for instance, when deleting a row that contains outliers. In this case, we first refine the

decomposition [100, 104], that is, compute an orthogonal matrix $\widehat{U}$ such that

$$\widehat{U} \begin{pmatrix} L & 0 \\ F & G \end{pmatrix} = \begin{pmatrix} \widetilde{L} & \widetilde{H} \\ 0 & \widetilde{G} \end{pmatrix}. \tag{4.35}$$

If it is still true that $\|a\| > 1$ even after the refinement, where $\widetilde{L}^T a = x$, the corrected semi-normal equation (CSNE) approach [18] (indicated by '+' in the first plot) is used for computing a with higher accuracy. It is essentially the same as that used by Park and Eldén [87] and is given by

$$\widetilde{L}y = a, \quad t = e_1 - X_j V_1 y$$

$$\widetilde{L}^T \delta a = V_1^T X_j^T t, \quad a = a + \delta a$$

$$\widetilde{L}\delta y = \delta a, \quad t = t - X_j V_1 \delta y, \quad \alpha = \|t\|$$

where $X_j^T = \begin{pmatrix} r & A_j^T \end{pmatrix}$, the j-th window matrix augmented with the row being deleted. Finally, restore the lower triangular form, that is, compute an orthogonal $\widetilde{V}$ such that

$$\begin{pmatrix} \widehat{L} & 0 \\ \widehat{F} & \widehat{G} \end{pmatrix} = \begin{pmatrix} \widetilde{L} & \widetilde{H} \\ 0 & \widetilde{G} \end{pmatrix} \widetilde{V}. \tag{4.36}$$

The CSNE approach was used in all three examples and most extensively in Example 4.3 when downdating a row with an outlier. However, the performance of our algorithm was less satisfactory for larger outliers. This is consistent with our error analysis, since for a very large outlier we would have ω in (4.26) very large.

The refinement steps in (4.35)-(4.36) require $k^2(n - k)$ Givens rotations, so that they become impractical when $k = \mathcal{O}(n^{\frac{1}{2}})$ or larger. As an alternative, we solve for a

from the equation

$$\left[L^T F^T \right] a = z \tag{4.37}$$

by solving

$$L^T a_B = z_m \tag{4.38}$$

$$\min_{a_N} \left\| \begin{pmatrix} -L^{-T} F^T \\ I \end{pmatrix} a_N + \begin{pmatrix} a_B \\ 0 \end{pmatrix} \right\|. \tag{4.39}$$

Then, we see that

$$a = \begin{pmatrix} a_B - L^{-T} F^T a_N \\ a_N \end{pmatrix} \tag{4.40}$$

is the minimum length solution to (4.37). (4.38) requires $\mathcal{O}(k^2)$ back substitution, and (4.39) can be approximated by a few steps of Lanczos algorithm since it is well-conditioned.

Table 4.1. Tracking $\|F\|_F$ and $\|G\|_F$ for the ULVD Procedure

Steps	Updating Formula	Flops
2	$\|F^{(2)}\|_F^2 = \|F^{(1)}\|_F^2 - \|f_1^{(1)}\|^2$	$\mathcal{O}(k)$
	$\|G^{(2)}\|_F^2 = \|G^{(1)}\|_F^2 - \{g_{11}^{(1)}\}^2 + \{g_{11}^{(2)}\}^2$	$\mathcal{O}(1)$
5	$\|F^{(3)}\|_F^2 = \|F^{(2)}\|_F^2 + \|g_1^{(3)}\|^2 - \|g_1^{(4)}\|^2$	$\mathcal{O}(p-k)$
	$\|G^{(4)}\|_F^2 = \|G^{(3)}\|_F^2 - \|g_1^{(3)}\|^2 + \|g_1^{(4)}\|^2$	$\mathcal{O}(1)$

Moreover, to prevent a from becoming too large, we track $\|F\|_F$ so that it remains under certain threshold, say, $\|L^{-1}\| \, \|F\|_F < 0.01$. This is similar to recommendations made by Fierro and Bunch [41, 42]. Only steps in Algorithm 4.1 that require to update $\|F\|_F$ are Steps 2 and 5. Table 4.1 shows how to update these quantities. Here, $g_1^{(i)}$ denotes the first column of $G^{(i)}, i = 1, 3, 4$.
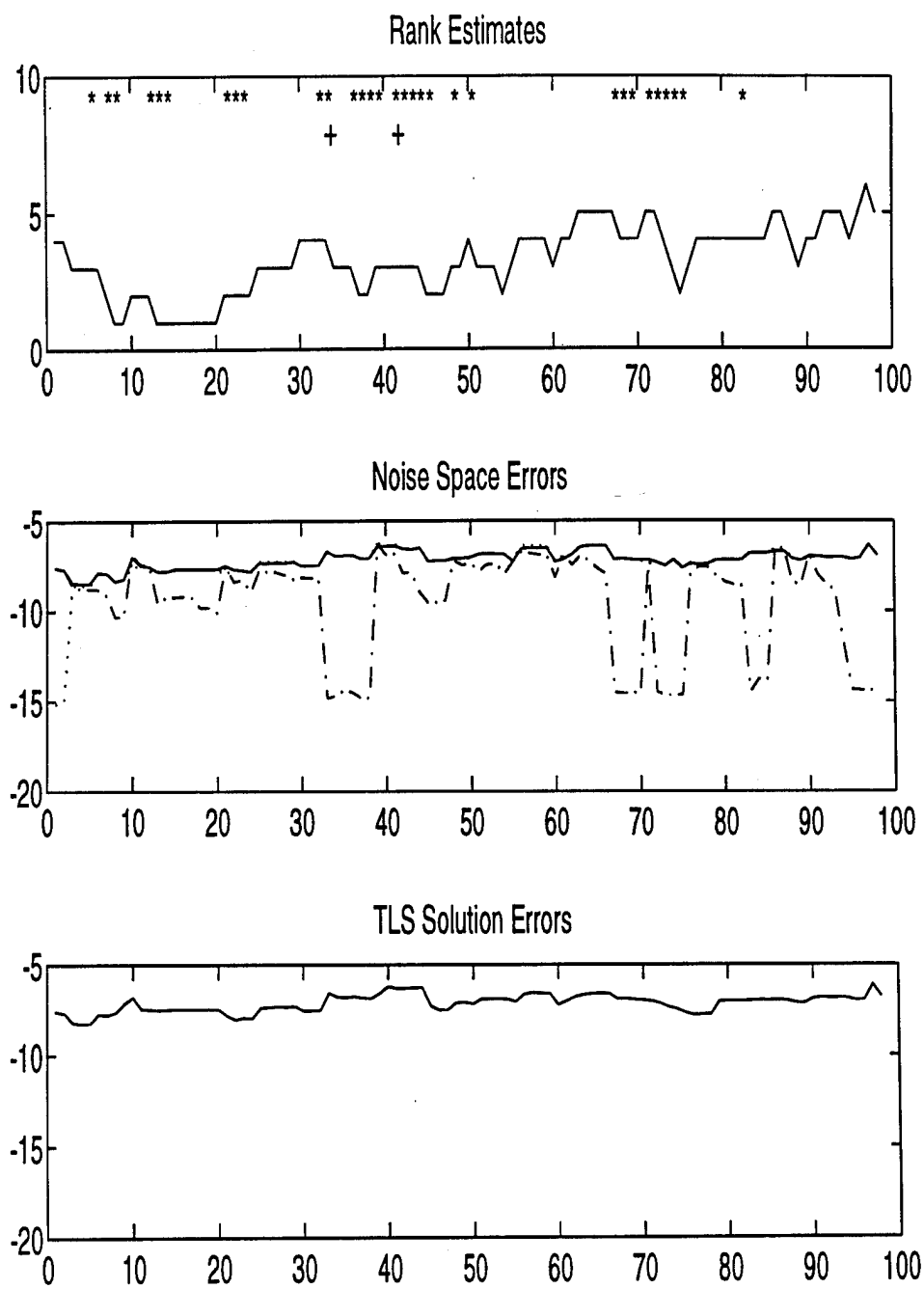
Fig. 4.3.   Example 4.1
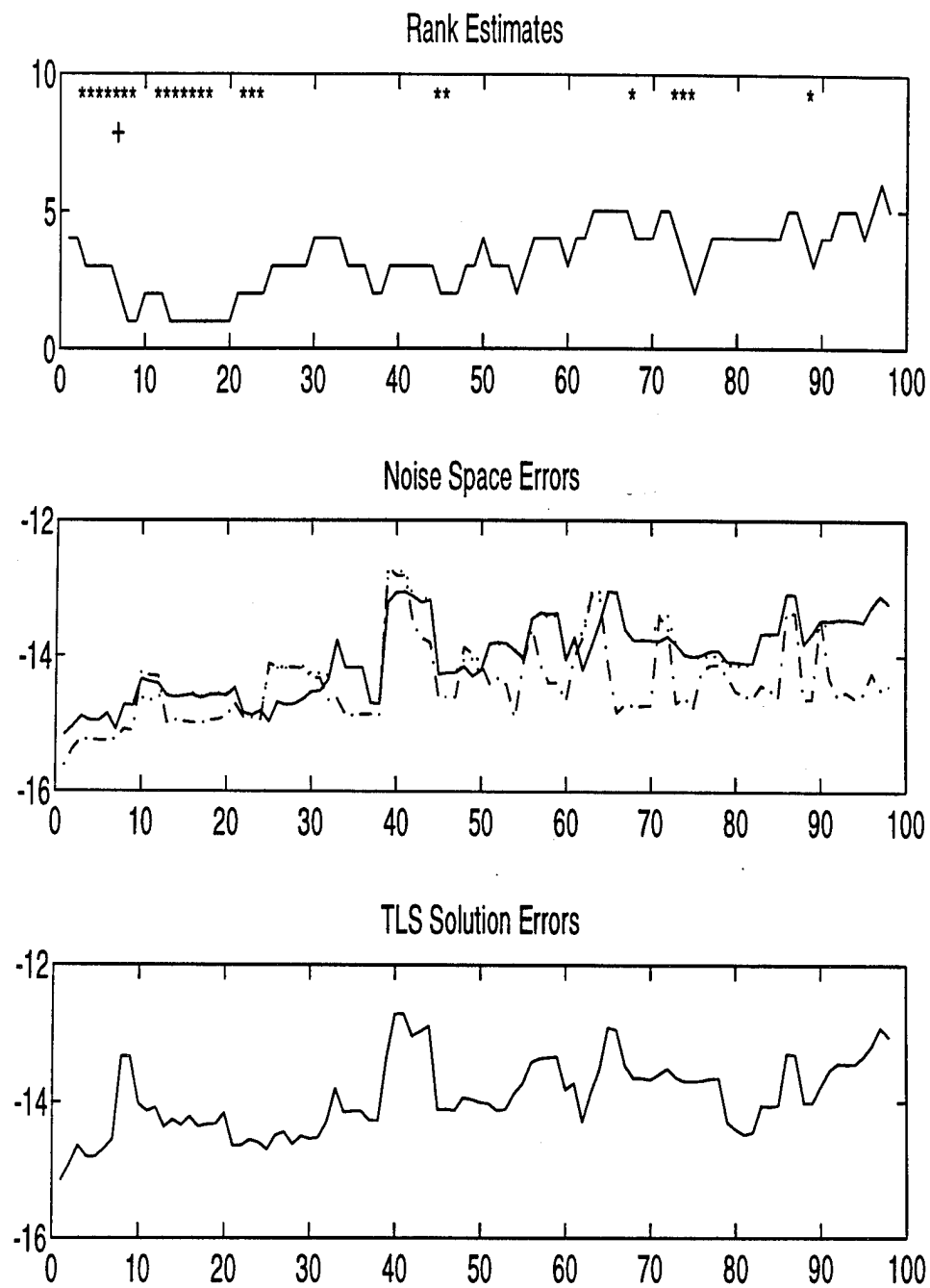
Fig. 4.4.   Example 4.2
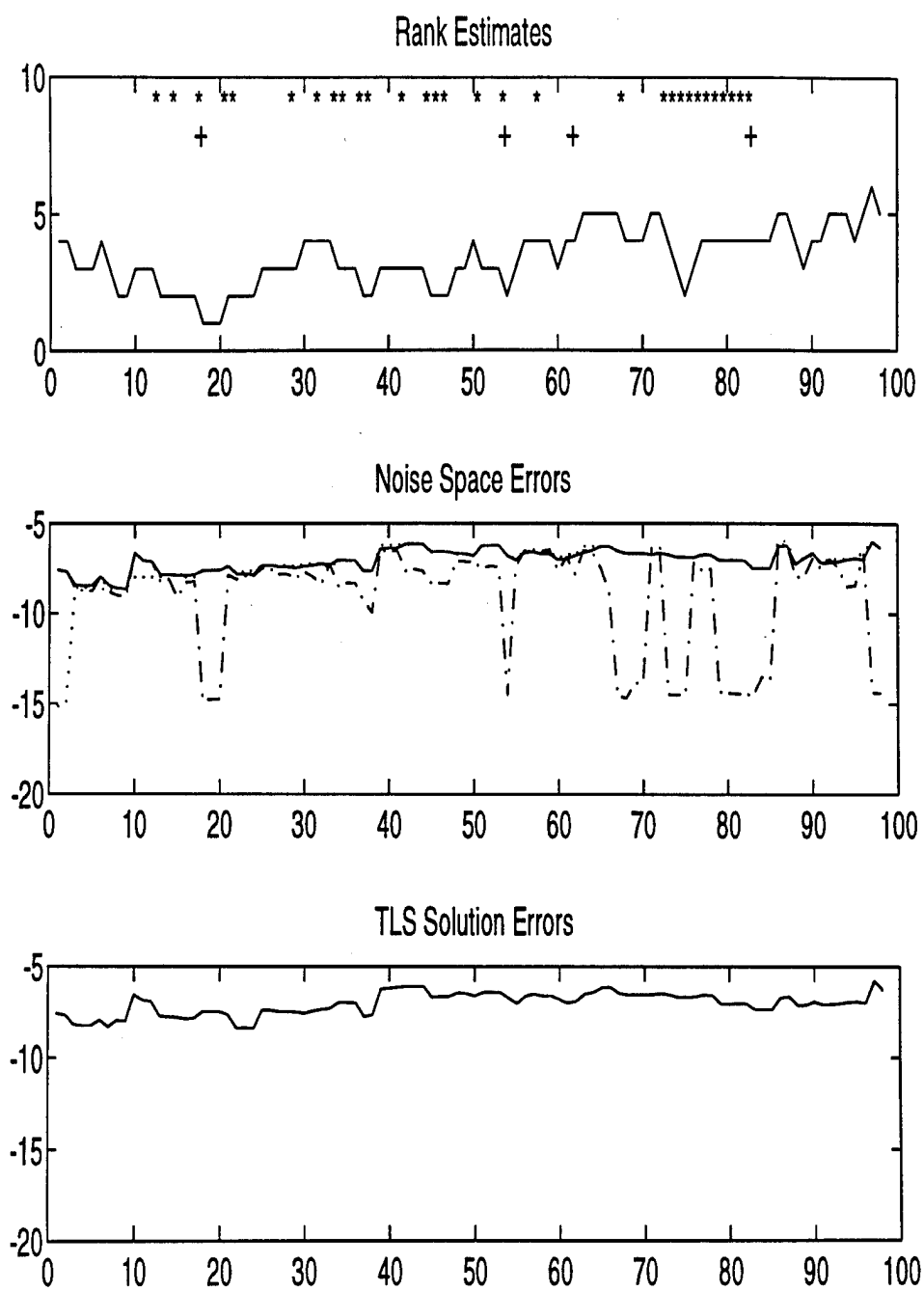
Fig. 4.5.   Example 4.3

Chapter 5

Rank Detection for Modifying the ULV Decomposition

5.1 Introduction

In Algorithm 4.1 the deflation steps are required to compute the numerical rank after a downdate. The deflation step usually involves some condition estimator, which can be a nuisance in some situations such as in parallel implementation. A survey of popular condition estimators is given in [61], and they all require $\mathcal{O}(k^2)$ flops, so that the entire process requires $\mathcal{O}(k^2)$ flops, where k is the dimension of L in (4.2). With some modification to Algorithm 4.1 we can often eliminate Step 7, the deflation step.

Furthermore, the modified algorithm offers an efficient way of tracking exact quantities of $\|L^{-1}\|_F$, $\|F\|_F$, and $\|G\|_F$, which give a significant information on the condition of the downdating problem. The experiments show that the computed subspaces are as good as can be expected, and no worse than those demonstrated in the previous chapter. The updating algorithm for the ULVD can be also implemented similarly with a slight modification to the downdating algorithm.

We propose our new ULVD updating/downdating algorithm in Section 5.2. Section 5.3 contains the rank detection method related to the new algorithm. In Section 5.4, we give numerical tests of our algorithm on the RTLS problems.

5.2 New Algorithm for ULVD Downdating

Our new algorithm reduces the downdating problem to a 2×2 downdating problem. As in Algorithm 4.1, we assume that $L^T L - xx^T$ is positive definite, and $\bar{U}$ is not accumulated.

ALGORITHM 5.1 (NEW PROCEDURE FOR ULVD DOWNDATING). Assume the terminology of Algorithm 4.1, and denote also that $l_{kk}^{(i)} = e_k^T [L^{(i)}] e_k$.

Step 1. Construct orthogonal matrices $\bar{U}_1, \bar{V}_1 \in \mathcal{R}^{k \times k}$ and $\bar{U}_2, \bar{V}_2 \in \mathcal{R}^{(p-k) \times (p-k)}$ such that

$$L^{(1)} = \bar{U}_1^T L \bar{V}_1, \quad \bar{V}_1^T x = \xi e_k, \quad \xi = \|x\| \tag{5.1}$$

$$G^{(1)} = \bar{U}_2^T G \bar{V}_2, \quad \bar{V}_2^T y = \rho e_1, \quad \rho = \|y\|. \tag{5.2}$$

Also, compute

$$F^{(1)} = \bar{U}_2^T F \bar{V}_1.$$

Update $\bar{V} \leftarrow \bar{V} \, \mathrm{diag}(\bar{V}_1, I_{n-k}) \; \mathrm{diag}(I_k, \bar{V}_2, I_{n-p}).$

Step 2. Find an orthogonal matrix $\bar{U}_3 \in \mathcal{R}^{(k+1) \times (k+1)}$ such that

$$\begin{pmatrix} L^{(2)} & h \\ 0 & g_{11}^{(2)} \end{pmatrix} = \bar{U}_3^T \begin{pmatrix} L^{(1)} & 0 \\ \{f_1^{(1)}\}^T & g_{11}^{(1)} \end{pmatrix}. \tag{5.3}$$

Define $F^{(2)} = (I - e_1 e_1^T) F^{(1)}.$

Step 3. Write

$$
L^{(2)} = \begin{array}{cc} \overset{k-1}{} & \overset{1}{} \end{array} \begin{pmatrix} L^{(2)}_{11} & 0 \\ \{w^{(2)}\}^T & l^{(2)}_{kk} \end{pmatrix} \begin{array}{c} k-1 \\ 1 \end{array},
$$

and find an orthogonal matrix $\bar{U}_4 \in \mathcal{R}^{k \times k}$ such that

$$
L^{(3)} = \begin{pmatrix} L^{(3)}_{11} & w^{(3)} \\ 0 & l^{(3)}_{kk} \end{pmatrix} = \bar{U}_4^T L^{(2)}. \tag{5.4}
$$

Compute also $h^{(1)} = \bar{U}_4^T h$.

Step 4. Use Algorithm 3.9 for 2×2 downdating:

$$
[l^{(4)}_{kk}, h^{(2)}_k, g^{(3)}_{11}] = \mathbf{down22}(l^{(3)}_{kk}, h^{(1)}_k, g^{(2)}_{11}, \beta_1, \beta_2)
$$

where we solve .

$$
\begin{pmatrix} l^{(3)}_{kk} & h^{(1)}_k \\ 0 & g^{(2)}_{11} \end{pmatrix}^T \begin{pmatrix} \tilde{\beta}_1 \\ \tilde{\beta}_2 \end{pmatrix} = \begin{pmatrix} \xi \\ \rho \end{pmatrix} \tag{5.5}
$$

and set

$$
\beta_1 = \tilde{\beta}_1, \quad \beta_2 = \min\{\tilde{\beta}_2, \sqrt{1 - \beta_1^2}\}. \tag{5.6}
$$

Step 5. Find an orthogonal matrix $\bar{V}_3 \in \mathcal{R}^{k \times k}$ such that

$$
L^{(5)} = \begin{pmatrix} L^{(5)}_{11} & 0 \\ \{w^{(5)}\}^T & l^{(5)}_{kk} \end{pmatrix} = L^{(4)} \bar{V}_3 \tag{5.7}
$$

where $L^{(4)}$ and $L^{(3)}$ differ only on the (k,k) entry. Also, compute

$$F^{(3)} = F^{(2)}\bar{V}_3.$$

Update $\bar{V} \leftarrow \bar{V}\,\text{diag}(\bar{V}_3, I_{n-k})$.

Step 6. Find an orthogonal matrix $\bar{V}_4 \in \mathcal{R}^{(k+1)\times(k+1)}$ such that

$$(L^{(6)} \quad 0) = (L^{(5)} \quad h^{(2)}e_1^T)\bar{V}_4 \tag{5.8}$$

$$(F^{(4)} \quad G^{(4)}) = (F^{(3)} \quad G^{(3)})\,\text{diag}(\bar{V}_4, I_{p-k-1}). \tag{5.9}$$

Update $\bar{V} \leftarrow \bar{V}\,\text{diag}(\bar{V}_4, I_{n-k-1})$. If $g_{11}^{(4)} \neq 0$, then

$$\bar{C} = \begin{pmatrix} \bar{L} & 0 & 0 \\ \bar{F} & \bar{G} & 0 \\ 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} L^{(6)} & 0 & 0 \\ F^{(4)} & G^{(4)} & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

and skip Step 7.

Step 7. If $g_{11}^{(4)} = 0$ then $f_1^{(4)} = 0$ also since it was formed from $g_{11}^{(3)}$ using $\bar{V}_4$. Thus

$$\begin{matrix} & \begin{matrix} k & \quad p-k \end{matrix} \\ \begin{matrix} k \\ p-k \end{matrix} & \begin{pmatrix} L^{(6)} & 0 \\ F^{(4)} & G^{(4)} \end{pmatrix} \end{matrix} = \begin{matrix} & \begin{matrix} k & \quad p-k \end{matrix} & \\ & \begin{pmatrix} \bar{L} & 0 \\ 0 & 0 \\ \bar{F} & \widetilde{G}^{(4)} \end{pmatrix} & \begin{matrix} k \\ 1 \\ p-k-1 \end{matrix} \end{matrix}$$

where $\widetilde{G}^{(4)}$ is a lower Hessenberg matrix. We then find an orthogonal matrix $\bar{V}_5 \in \mathcal{R}^{(p-k) \times (p-k)}$ and an orthogonal permutation matrix $\bar{U}_5 \in \mathcal{R}^{(p-k) \times (p-k)}$ such that

$$\bar{U}_5^T G^{(4)} \bar{V}_5 = \begin{pmatrix} \bar{G} & 0 \\ 0 & 0 \end{pmatrix} \begin{matrix} p-k-1 \\ 1 \end{matrix} \ .$$

Update $\bar{V} \leftarrow \bar{V} \operatorname{diag}(I_k, \bar{V}_5, I_{n-p})$. Thus,

$$\bar{C} = \begin{array}{c} \begin{matrix} k & p-k-1 & n-p+1 \end{matrix} \\ \begin{pmatrix} \bar{L} & 0 & 0 \\ \bar{F} & \bar{G} & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{matrix} k \\ p-k-1 \\ n-p+1 \end{matrix} \end{array}$$

Step 8. Update the numerical rank of $\bar{L}$, and make appropriate adjustments to $\bar{F}$ and $\bar{G}$ (a procedure for the rank detection will be described in the next section).

Algorithm 5.1 requires $12k^2 + 6(p-k)^2 + 24k(p-k) + \mathcal{O}(p)$ flops for Steps 1-6 and $4k^2 + \mathcal{O}(k)$ flops for Step 7. When V is modified, additional $6np + 12nk$ flops will be required. Fig. 5.1–5.2 depict the reduction steps for Algorithm 5.1. As in Algorithm 4.1, a set of $\overset{*}{\rightarrow}$ denotes the downdating Step 4, and Step 7 is illustrated in Fig. 4.2.

REMARK 5.1. Note that in (5.5)–(5.6) we incorporate "additive noise" to compute α as similarly done in Step 4 of Algorithm 4.1. As shown in Chapter 4, the algorithm can be made consistent with this additive noise in the absence of rounding error. Proposition 4.2 shows this important consistency property.

REMARK 5.2. Algorithm 5.1 has a few advantages over Algorithm 4.1. Algorithm 5.1 does not incorporate the deflation process to estimate the numerical rank as often as

$$
\begin{pmatrix}
\hat{x} & x & x & y & y & \hat{y} \\
l & & & & & \\
l & l & & & & \\
l & l & l & & & \\
f & f & f & g & & \\
f & f & f & g & g & \\
f & f & f & g & g & g
\end{pmatrix}
\rightarrow
\begin{pmatrix}
0 & x & x & y & y & 0 \\
l & \hat{l} & & & & \\
l & l & & & & \\
l & l & l & & & \\
f & f & f & g & & \\
f & f & f & g & g & \hat{g} \\
f & f & f & g & g & g
\end{pmatrix}
\begin{pmatrix}
0 & \hat{x} & x & y & \hat{y} & 0 \\
l & & & & & \\
l & l & & & & \\
l & l & l & & & \\
f & f & f & g & & \\
f & f & f & g & g & \\
f & f & f & g & g & g
\end{pmatrix}
$$

$$
\rightarrow
\begin{pmatrix}
0 & 0 & \xi & \rho & 0 & 0 \\
l & & & & & \\
l & l & \hat{l} & & & \\
l & l & l & & & \\
f & f & f & g & \hat{g} & \\
f & f & f & g & g & \\
f & f & f & g & g & g
\end{pmatrix}
\rightarrow
\begin{pmatrix}
0 & 0 & \xi & \rho & 0 & 0 \\
l & & & & & \\
l & l & & & & \\
l & l & l & & & \\
f & f & \hat{f} & g & & \\
f & f & f & g & g & \\
f & f & f & g & g & g
\end{pmatrix}
\rightarrow
\begin{pmatrix}
0 & 0 & \xi & \rho & 0 & 0 \\
l & & & & & \\
l & l & & & & \\
l & l & l & g & & \\
f & \hat{f} & 0 & g & & \\
f & f & f & g & g & \\
f & f & f & g & g & g
\end{pmatrix}
$$

$$
\rightarrow
\begin{pmatrix}
0 & 0 & \xi & \rho & 0 & 0 \\
l & & & & & \\
l & l & & g & & \\
l & l & l & g & & \\
\hat{f} & 0 & 0 & g & & \\
f & f & f & g & g & \\
f & f & f & g & g & g
\end{pmatrix}
\rightarrow
\begin{pmatrix}
0 & 0 & \xi & \rho & 0 & 0 \\
l & & & g & & \\
l & l & & g & & \\
l & \hat{l} & l & g & & \\
0 & 0 & 0 & g & & \\
f & f & f & g & g & \\
f & f & f & g & g & g
\end{pmatrix}
\rightarrow
\begin{pmatrix}
0 & 0 & \xi & \rho & 0 & 0 \\
l & & & g & & \\
l & l & l & g & & \\
\hat{l} & 0 & l & g & & \\
0 & 0 & 0 & g & & \\
f & f & f & g & g & \\
f & f & f & g & g & g
\end{pmatrix}
$$

$$
\begin{pmatrix}
0 & 0 & \xi & \rho & 0 & 0 \\
l & & & g & & \\
l & l & l & g & & \\
0 & 0 & l & g & & \\
0 & 0 & 0 & g & & \\
f & f & f & g & g & \\
f & f & f & g & g & g
\end{pmatrix}
$$

Fig. 5.1. Improved Reduction Steps 1–3 for Downdating the ULVD

$$
\begin{array}{l}
\overset{*}{\rightarrow} \\
\\
\\
\overset{*}{\rightarrow} \\
\overset{*}{\rightarrow} \\
\\
\end{array}
\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 \\
l & & l & g & & \\
l & l & l & g & & \\
0 & 0 & l & g & & \\
0 & 0 & 0 & g & & \\
f & f & f & g & g & \\
f & f & f & g & g & g
\end{pmatrix}
\quad
\begin{array}{cc} \downarrow & \ \downarrow \end{array}
\begin{pmatrix}
x & x & x & y & y & y \\
l & & \widehat{l} & g & & \\
l & l & l & g & & \\
0 & 0 & l & g & & \\
0 & 0 & 0 & g & & \\
f & f & f & g & g & \\
f & f & f & g & g & g
\end{pmatrix}
\quad
\begin{array}{cc} \downarrow & \downarrow \end{array}
\begin{pmatrix}
x & x & x & y & y & y \\
l & & & g & & \\
l & l & \widehat{l} & g & & \\
l & 0 & l & g & & \\
0 & 0 & 0 & g & & \\
f & f & f & g & g & \\
f & f & f & g & g & g
\end{pmatrix}
$$

$$
\begin{array}{cc} \downarrow & \ \downarrow \end{array}
\begin{pmatrix}
x & x & x & y & y & y \\
l & & & \widehat{g} & & \\
l & l & & g & & \\
l & l & l & g & & \\
0 & 0 & 0 & g & & \\
f & f & f & g & g & \\
f & f & f & g & g & g
\end{pmatrix}
\quad
\begin{array}{cc} \downarrow & \ \downarrow \end{array}
\begin{pmatrix}
x & x & x & y & y & y \\
l & & & & & \\
l & l & & \widehat{g} & & \\
l & l & l & g & & \\
f & 0 & 0 & g & & \\
f & f & f & g & g & \\
f & f & f & g & g & g
\end{pmatrix}
\quad
\begin{array}{cc} \downarrow & \downarrow \end{array}
\begin{pmatrix}
x & x & x & y & y & y \\
l & & & & & \\
l & l & & & & \\
l & l & l & \widehat{g} & & \\
f & f & 0 & g & & \\
f & f & f & g & g & \\
f & f & f & g & g & g
\end{pmatrix}
$$

$$
\begin{pmatrix}
\bar{x} & \bar{x} & \bar{x} & \bar{y} & \bar{y} & \bar{y} \\
l & & & & & \\
l & l & & & & \\
l & l & l & & & \\
f & f & f & g & & \\
f & f & f & g & g & \\
f & f & f & g & g & g
\end{pmatrix}
$$

Fig. 5.2. Improved Reduction Steps 4–6 for Downdating the ULVD

Algorithm 4.1. It turns out that after a downdate, $\bar{l}_{kk}$, the (k, k) entry of $\bar{L}$ often gives a very good approximation to $\sigma_{\min}(\bar{L})$. Steps 3 and 5 offer this benefit with some extra work, namely, $2(k-1)$ Givens rotations.

REMARK 5.3. The algorithm can be also used for updating the ULVD just by replacing Step 4 with the Algorithm 3.8, which requires only 12 flops.

5.3   Rank Detection

5.3.1   Bounding $\|L^{-1}\|$

In this section, we show that in Step 8 we can often determine the numerical rank of A without the deflation process. We start with stating a simple lemma that bounds the 2-norm of a block matrix in terms of 2-norms of its blocks.

LEMMA 5.1 ([9, LEMMA 3.3]). *Let M and $\widehat{M}$ be the $s \times s$ block matrices,*

$$
M = \begin{pmatrix} M_{11} & M_{12} & \cdots & M_{1s} \\ \vdots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ M_{s1} & M_{s2} & \cdots & M_{ss} \end{pmatrix}, \quad \widehat{M} = \begin{pmatrix} \|M_{11}\| & \|M_{12}\| & \cdots & \|M_{1s}\| \\ \vdots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ \|M_{s1}\| & \|M_{s2}\| & \cdots & \|M_{ss}\| \end{pmatrix}.
$$

Then $\|M\| \leq \|\widehat{M}\|$.

Proof. See the proof in [9]. □

In fact, Lemma 5.1 holds for any consistent norms. A straightforward application of this lemma results in the following lemma.

LEMMA 5.2. *Let*

$$L = \begin{pmatrix} L_{11} & w \\ 0 & \gamma \end{pmatrix} \begin{matrix} p \\ 1 \end{matrix} \qquad \begin{matrix} p & 1 \end{matrix}$$

(5.10)

where L_{11} is nonsingular and $\gamma \neq 0$. Then,

$$\|L^{-1}\| \leq \left\| \begin{pmatrix} \|L_{11}^{-1}\| & \gamma^{-1}\|L_{11}^{-1}w\| \\ 0 & \gamma^{-1} \end{pmatrix} \right\|.$$

(5.11)

Proof. It is easy to verify that

$$L^{-1} = \begin{pmatrix} L_{11}^{-1} & -\gamma^{-1}L_{11}^{-1}w \\ 0 & \gamma^{-1} \end{pmatrix}.$$

(5.12)

Then taking norms and using Lemma 5.1 yield (5.11). □

The following lemma shows the effect of Steps 3 and 5 of Algorithm 5.1 on $l_{kk}^{(i)}$, (k, k) entry of $L^{(i)}$, $i = 2, 4$.

LEMMA 5.3. *Let L be defined in (5.10). Suppose*

$$\tilde{L} = \begin{pmatrix} \tilde{L}_{11} & 0 \\ \tilde{w}^T & \tilde{\gamma} \end{pmatrix} = LQ$$

(5.13)

where Q is orthogonal. Then,

$$\tilde{\gamma}^{-1} = \gamma^{-1}(1 + \|L_{11}^{-1}w\|^2)^{1/2}.$$

(5.14)

Proof. From (5.13) we see that

$$\widetilde{L}^{-1} = \begin{pmatrix} \widetilde{L}_{11}^{-1} & 0 \\ -\widetilde{\gamma}^{-1}\widetilde{w}^T\widetilde{L}_{11}^{-1} & \widetilde{\gamma}^{-1} \end{pmatrix} = Q^T L^{-1}$$

where the expression for L^{-1} is given in (5.12). Thus we have

$$\widetilde{L}^{-T}\widetilde{L}^{-1} = L^{-T}L^{-1}$$

Comparing both sides yields (5.14). □

Combining Lemma 5.2 and 5.3 gives the following result.

LEMMA 5.4. *Assume the hypothesis of Lemma 5.3. Then*

$$\|L^{-1}\| \leq \psi(\|L_{11}^{-1}\|, \widetilde{\gamma}^{-1}, \gamma^{-1}), \tag{5.15}$$

where

$$\psi(x, y, z) = \left[\frac{1}{2}(x^2 + y^2) + \frac{1}{2}\sqrt{(x^2 + y^2)^2 - 4x^2z^2}\right]^{\frac{1}{2}}. \tag{5.16}$$

Proof. It is easy to show that for any 2×2 upper triangular matrix,

$$S = \begin{pmatrix} a & b \\ 0 & c \end{pmatrix},$$

we have

$$\sigma_1^2(S) = \frac{1}{2}(a^2 + b^2 + c^2) + \frac{1}{2}\sqrt{(a^2 + b^2 + c^2)^2 - 4a^2c^2}. \qquad (5.17)$$

Let $a = \|L_{11}^{-1}\|$, $b = -\gamma^{-1}\|L_{11}^{-1}w\|$, $c = \gamma^{-1}$. Then by Lemma 5.3, we see that $b^2 + c^2 = \tilde{\gamma}^{-2}$, and by Lemma 5.2, $\|L^{-1}\| \leq \sigma_1(S)$. Hence, taking a square root in (5.17) proves the lemma. $\square$

The following lemma states necessary conditions to ensure a correct rank estimation for the ULVD.

LEMMA 5.5. *Let C be defined in (4.2), and let $\eta = \|FL^{-1}\|$. Suppose*

$$\|L^{-1}\| \leq tol^{-1}, \quad \|G\| \, (1 + \eta) \leq tol, \quad \eta < 1. \qquad (5.18)$$

Then,

$$\sigma_k(C) \geq tol \geq \sigma_{k+1}(C). \qquad (5.19)$$

Proof. By singular value interlacing property [63, Theorem 7.3.9], it is easy to show that

$$\sigma_k(C) \geq \sigma_{\min}(L) = \|L^{-1}\|^{-1}, \quad \sigma_{k+1}(C) \leq \sigma_1(G) = \|G\|. \qquad (5.20)$$

Thus,

$$\sigma_k(C) \geq tol \geq tol \cdot \frac{1}{1 + \eta} \geq \|G\| \geq \sigma_{k+1}(C).$$

This proves the lemma. $\square$

Thus, the decomposition always remains rank-revealing as long as the conditions in (5.18) are satisfied. Note that these conditions are not enforceable unless there is a

reasonable gap in the spectrum, but the conditions for its existence are weaker than the RRQR conditions of Hong and Pan [62].

A cluster of singular values around *tol* would cause one of the conditions in (5.18) to be violated, and the rank to be underestimated. Since we keep tracking $\|F\|_F$ and $\|G\|_F$ automatically as a part of the algorithm as illustrated in Section 5.3.3, we will always know when this happens.

The quantities in the above lemma cannot be tracked efficiently using our algorithm. However, some good bounds can be tracked. That leads to the following theorem that uses only computed quantities.

THEOREM 5.1. *Consider the downdating procedure in Algorithm 5.1 and the related updating procedure given by Remark 3.5. Let C be the matrix before updating (downdating) and partitioned as in (4.2), and let L have the condition estimate $\kappa \approx \|L^{-1}\|^{-1}$ such that*

$$\sigma_k(C) \geq \kappa \geq tol \geq \sigma_{k+1}(C).$$

Let $\bar{C}$ be the matrix after updating (downdating) and partitioned according to

$$\bar{C} = \begin{pmatrix} \bar{L} & 0 & 0 \\ \bar{F} & \bar{G} & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{matrix} s \\ p-s \\ n-p \end{matrix} \qquad (5.21)$$

where $s = k + 1$ for updating and $s = k$ for downdating. Define

$$\bar{\kappa}^{-1} = \psi(\kappa^{-1}, |\bar{l}_{ss}|^{-1}, |l_{ss}|^{-1}) \qquad (5.22)$$

where $\psi(x, y, z)$ is the function defined in (5.16).

If

    (i) $\bar{\kappa} \geq tol$

    (ii) $\|\bar{F}\|_F / \bar{\kappa} = \bar{\eta} < 1$

    (iii) $\|\bar{G}\|_F \leq tol(1 + \bar{\eta})^{-1}$

then

$$\sigma_s(\bar{C}) \geq tol \geq \sigma_{s+1}(\bar{C}). \tag{5.23}$$

If

    (iv) $\bar{\kappa} \geq tol$

    (v) $\kappa^{-1} \sqrt{\|\bar{F}\|_F^2 + \|e_s^T \bar{L}\|^2 - \|\bar{F}e_s\|^2} = \tilde{\eta} < 1$

    (vi) $\sqrt{\|\bar{G}\|_F^2 + \|\bar{F}e_s\|^2 + |\bar{l}_{ss}|^2} \leq tol(1 + \tilde{\eta})^{-1}$

then

$$\sigma_{s-1}(\bar{C}) \geq tol \geq \sigma_s(\bar{C}). \tag{5.24}$$

Proof. Using the fact that for any matrix A, $\|A\| \leq \|A\|_F$, it is simple to show that the conditions (i)–(iii) satisfy the hypothesis of Lemma 5.5. Thus, (5.23) immediately follows.

Let

$$\tilde{C} = \begin{array}{c} \\ \\ \end{array} \overset{\displaystyle \begin{array}{ccc} s-1 & p-s+1 & n-p \end{array}}{\left(\begin{array}{ccc} \tilde{L} & 0 & 0 \\ \tilde{F} & \tilde{G} & 0 \\ 0 & 0 & 0 \end{array} \right)} \begin{array}{c} s-1 \\ p-s+1 \\ n-p \end{array} ,$$

where

$$\bar{L} \;=\; \begin{matrix} \scriptstyle s-1 & \scriptstyle 1 \\ \begin{pmatrix} \tilde{L} & 0 \\ \bar{w}^T & \bar{l}_{ss} \end{pmatrix} & \begin{matrix} \scriptstyle s-1 \\ \scriptstyle 1 \end{matrix} \end{matrix} \;, \qquad \bar{w} = \tilde{L}^T e_s$$

$$\tilde{F} \;=\; \begin{pmatrix} \bar{w}^T \\ \widehat{F} \end{pmatrix}, \qquad \bar{F} = \begin{pmatrix} \widehat{F} & \bar{f}_s \end{pmatrix}, \qquad \bar{f}_s = \bar{F} e_s$$

$$\tilde{G} \;=\; \begin{matrix} \scriptstyle 1 & \scriptstyle p-s \\ \begin{pmatrix} \bar{l}_{ss} & 0 \\ \bar{f}_s & \bar{G} \end{pmatrix} & \begin{matrix} \scriptstyle 1 \\ \scriptstyle p-s \end{matrix} \end{matrix} \;.$$

Then, it is simple to verify that

$$\|\tilde{F}\|_F^2 = \|\bar{F}\|_F^2 + \|w\|^2 - \|\bar{f}_s\|^2$$
$$\|\tilde{G}\|_F^2 = \|\bar{G}\|_F^2 + \|\bar{f}_s\|^2 + |\bar{l}_{ss}|^2$$

Again, the conditions (v)–(vi) ensure that $\|\tilde{F}\|_F$ and $\|\tilde{G}\|_F$ satisfy the hypothesis of Lemma 5.5. Hence, (5.24) also follows. □

The conditions (iii) and (vi) make certain that there exists a reasonable gap in the spectrum. When the conditions (i)–(iii) are satisfied, updating results in a rank increase. Similarly, when the conditions (iv)–(vi) are met, downdating results in a rank decrease. From the theorem we see the importance of keeping $\|F\|$ as small as possible. Several ways to do this are proposed in [41, 42, 104].

The theorem still holds even if κ is replaced by $\kappa \approx \|\bar{L}^{-1}\|_F$ in (5.22). We can explicitly compute $\|\bar{L}^{-1}\|_F$ in $\mathcal{O}(n^2)$ flops as described in the following section. In some cases, this may be a more pessimistic bound than the one in Theorem 5.1.

We should note that there is still a possibility that $\bar{\kappa}$ could be any underestimate of $\sigma_k(\bar{L})$. Thus, if the conditions of this theorem are not satisfied, then the condition of $\bar{L}$ will still have to be estimated. When the conditions (iii) or (vi) are violated, indicating that there exists little gap between them, one must refactor or redefine *tol*. If none of these work, one can conclude that the ULVD may not be suitable for tracking subspaces for the problem under consideration.

One of the factors that might affect the rank estimation using this scheme is incorrect rank estimate from the previous update/downdate. This problem, however, can be solved by computing the initial factorization using the SVD for an accurate rank estimation. Although the SVD is more expensive to compute than the ULVD, the cost will become negligible when amortized over the cost at the subsequent updating and downdating steps.

Next two sections will show how to track efficiently the quantities, $\|L^{-1}\|_F$, $\|F\|_F$, and $\|G\|_F$.

5.3.2 Tracking $\|L^{-1}\|_F$

We begin with the following lemma.

LEMMA 5.6. *Assume the hypothesis and terminology of Lemma 5.3. Then,*

$$\|L^{-1}\|_F^2 = \|L_{11}^{-1}\|_F^2 + \frac{1}{\tilde{\gamma}^2} \tag{5.25}$$

Proof. From (5.12) we see that

$$\|L^{-1}\|_F^2 = \|L_{11}^{-1}\|_F^2 + \frac{1 + \|L^{-1}w\|^2}{\gamma^2}.$$

Then, (5.25) follows directly from Lemma 5.3. □

Because of orthogonal invariance of $\|\cdot\|_F$ we observe that

$$\|\{L^{(1)}\}^{-1}\|_F = \|L^{-1}\|_F$$

$$\|\{L^{(3)}\}^{-1}\|_F = \|\{L^{(2)}\}^{-1}\|_F$$

$$\|\{L^{(5)}\}^{-1}\|_F = \|\{L^{(4)}\}^{-1}\|_F$$

where we used the terminology of Algorithm 5.1. Thus, the steps in Algorithm 5.1 that we need to consider updating $\|L^{-1}\|_F$ are Steps 2, 4, and 6. The following lemmata give the formula for each step.

LEMMA 5.7. *Assume the terminology of Algorithm 5.1. Then*

$$\|\{L^{(4)}\}^{-1}\|_F^2 = \|\{L^{(3)}\}^{-1}\|_F^2 + \frac{\{l_{kk}^{(3)}\}^2 - \{l_{kk}^{(4)}\}^2}{\left[l_{kk}^{(3)}l_{kk}^{(5)}\right]^2} \qquad (5.26)$$

Proof. Since

$$\|\{L^{(3)}\}^{-1}\|_F^2 = \|\{L_{11}^{(3)}\}^{-1}\|_F^2 + \frac{1 + \|\{L_{11}^{(3)}\}^{-1}w^{(3)}\|^2}{\{l_{kk}^{(3)}\}^2} \qquad (5.27)$$

$$\|\{L^{(4)}\}^{-1}\|_F^2 = \|\{L_{11}^{(3)}\}^{-1}\|_F^2 + \frac{1 + \|\{L_{11}^{(3)}\}^{-1}w^{(3)}\|^2}{\{l_{kk}^{(4)}\}^2} \qquad (5.28)$$

we see that

$$\|\{L^{(4)}\}^{-1}\|_F^2 \;=\; \|\{L^{(3)}\}^{-1}\|_F^2 - \frac{1 + \|\{L_{11}^{(3)}\}^{-1} w^{(3)}\|^2}{\{l_{kk}^{(3)}\}^2} \tag{5.29}$$

$$+ \; \frac{1 + \|\{L_{11}^{(3)}\}^{-1} w^{(3)}\|^2}{\{l_{kk}^{(4)}\}^2},$$

By Lemma 5.3 we obtain

$$\frac{1 + \|\{L_{11}^{(3)}\}^{-1} w^{(3)}\|^2}{\{l_{kk}^{(4)}\}^2} = \frac{1}{\{l_{kk}^{(5)}\}^2}, \tag{5.30}$$

Substituting (5.30) into (5.29) yields the result. □

Since $l_{kk}^{(5)}$ is not available at Step 4, and $\|\{L^{(5)}\}^{-1}\|_F = \|\{L^{(4)}\}^{-1}\|_F$, we update this at Step 5.

LEMMA 5.8. *Assume the terminology of Algorithm 5.1. Then*

$$\|\{L^{(2)}\}^{-1}\|_F^2 \;=\; \|\{L^{(1)}\}^{-1}\|_F^2 - \frac{\|c\|^2}{\{g_{11}^{(2)}\}^2} \tag{5.31}$$

$$\|\{L^{(6)}\}^{-1}\|_F^2 \;=\; \|\{L^{(5)}\}^{-1}\|_F^2 - \frac{\|d\|^2}{\{g_{11}^{(4)}\}^2} \tag{5.32}$$

where $L^{(2)} c = h$, *and* $\{L^{(6)}\}^T d = f_1^{(4)}$.

Proof. Let

$$\widehat{L}^{(1)} = \begin{pmatrix} L^{(1)} & 0 \\ \{f_1^{(1)}\}^T & g_{11}^{(1)} \end{pmatrix}, \quad \widehat{L}^{(2)} = \begin{pmatrix} L^{(2)} & h \\ 0 & g_{11}^{(2)} \end{pmatrix} = \bar{U}_3^T \widehat{L}^{(1)}$$

Then we obtain

$$\|\{\widehat{L}^{(1)}\}^{-1}\|_F^2 \;=\; \|\{L^{(1)}\}^{-1}\|_F^2 + \frac{1 + \|\{L^{(1)}\}^{-T} f_1^{(1)}\|^2}{\{g_{11}^{(1)}\}^2} \tag{5.33}$$

$$\|\{\widehat{L}^{(2)}\}^{-1}\|_F^2 \;=\; \|\{L^{(2)}\}^{-1}\|_F^2 + \frac{1 + \|\{L^{(2)}\}^{-1} h\|^2}{\{g_{11}^{(2)}\}^2} \tag{5.34}$$

By orthogonal invariance, we see that $\|\{L^{(1)}\}^{-1}\|_F = \|\{L^{(2)}\}^{-1}\|_F$, so that

$$\|\{L^{(2)}\}^{-1}\|_F^2 \;=\; \|\{L^{(1)}\}^{-1}\|_F^2 + \frac{1 + \|\{L^{(1)}\}^{-T} f_1^{(1)}\|^2}{\{g_{11}^{(1)}\}^2} \tag{5.35}$$

$$\qquad - \frac{1 + \|\{L^{(2)}\}^{-1} h\|^2}{\{g_{11}^{(2)}\}^2}$$

By Lemma 5.3, we see that

$$\frac{1 + \|\{L^{(1)}\}^{-T} f_1^{(1)}\|^2}{\{g_{11}^{(1)}\}^2} = \frac{1}{\{g_{11}^{(2)}\}^2}. \tag{5.36}$$

Substituting (5.36) into (5.35) yields (5.31). The proof of (5.32) is similar. □

Computing $\|\{L^{(2)}\}^{-1}\|_F$ and $\|\{L^{(6)}\}^{-1}\|_F$ requires $\mathcal{O}(k^2)$ flops for forward and backward substitutions, respectively, and $\|\{L^{(4)}\}^{-1}\|_F$, $\mathcal{O}(1)$ flops.

Finally, when there is a rank change in Step 8, we use the relation,

$$\|\bar{L}^{-1}\|_F^2 = \|\bar{L}_{11}^{-1}\|_F^2 + \frac{1 + \|\bar{w}^T \bar{L}_{11}^{-1}\|^2}{\{\bar{l}_{kk}\}^2}$$

where

$$\bar{L} = \begin{pmatrix} \bar{L}_{11} & 0 \\ \bar{w}^T & \bar{l}_{kk} \end{pmatrix}.$$ (5.37)

Thus, it takes $\mathcal{O}(k^2)$ back substitution.

All of these formulae also work for updating as well as downdating for obvious reasons. Updating and downdating procedures differ only on Step 4 when updating and downdating 2×2 matrices. Thus, $l_{kk}^{(4)}$ is computed in two different ways. Therefore, the formula (5.26) can be also used with the updating procedure.

5.3.3   Tracking $\|F\|_F$ and $\|G\|_F$

For completeness we show how to update the $\|F\|_F$ and $\|G\|_F$ in Table 5.1 although it was partially described in [13]. Here, we denote $g_1^{(i)} = [G^{(i)}]e_1, i = 3, 4$. Note here that having calculated $\|g^{(3)}\|$ and $\|g^{(4)}\|$ to compute $\|F^{(4)}\|_F$, computing $\|G^{(4)}\|_F$ only requires $\mathcal{O}(1)$ flops. As in tracking $\|L^{-1}\|_F$, all of the formulae in Table 5.1 also work for updating as well as downdating.

Table 5.1.   Tracking $\|F\|_F$ and $\|G\|_F$ for the Improved ULVD Procedure

Steps	Updating Formula	Flops
2	$\|F^{(2)}\|_F^2 = \|F^{(1)}\|_F^2 - \|f_1^{(1)}\|^2$	$\mathcal{O}(k)$
	$\|G^{(2)}\|_F^2 = \|G^{(1)}\|_F^2 - \{g_{11}^{(1)}\}^2 + \{g_{11}^{(2)}\}^2$	$\mathcal{O}(1)$
4	$\|G^{(3)}\|_F^2 = \|G^{(2)}\|_F^2 - \{g_{11}^{(2)}\}^2 + \{g_{11}^{(3)}\}^2$	$\mathcal{O}(1)$
6	$\|F^{(4)}\|_F^2 = \|F^{(3)}\|_F^2 + \|g_1^{(3)}\|^2 - \|g_1^{(4)}\|^2$	$\mathcal{O}(p-k)$
	$\|G^{(4)}\|_F^2 = \|G^{(3)}\|_F^2 - \|g_1^{(3)}\|^2 + \|g_1^{(4)}\|^2$	$\mathcal{O}(1)$

5.4 Numerical Examples

In this section, we present a few examples from numerical experiments. These tests were performed using MATLAB on a SPARCstation 5 in IEEE Standard double precision with machine precision $\approx 10^{-16}$. As in Chapter 4 the algorithm employs the sliding window technique from signal processing.

At each step of the sliding window method with the window size m_0, an $m_0 \times n$ data matrix is constructed from an $m \times n$ observation matrix A by adding a new row to the data matrix in the previous window and deleting the oldest row from it. In step j, the row $m_0 + j$ of the observation matrix is added and the row j is deleted, giving the data matrix A_j. The ULVD of the initial window matrix A_0, which consists of the first m_0 rows of A, can be obtained by computing its SVD.

Then Algorithm 5.1 takes the lower triangular matrix (middle part of the decomposition), the orthogonal matrix (right part) as initial input and the modifying vector r, and successively modifies these matrices at every window step. The vector $z = V^T r$ is computed at the beginning of each window step.

We tested our algorithms in the context of the total least squares (TLS) problems. See Section 2.7 for details. We use the TLS solutions from the Jacobi SVD as reference in checking the accuracy of the solution and rank estimates of our algorithms.

Fig. 5.3-5.5 show the rank estimates by Algorithms 4.1 and 5.1. The horizontal axis represents the window steps and the vertical axis the numerical rank of the window matrix.

The distance between the subspaces is given in the next plot using the Definition 2.3. Let

$$A_j = Y_j \Sigma_j W_j^T, \quad W_j = (W_{j1} \quad W_{j2})$$

be the SVD of A_j computed by the one-sided Jacobi method at step j. Let

$$A_j = U_j^{(i)} C_j^{(i)} \{V_j^{(i)}\}^T, \quad V_j^{(i)} = (V_{j1}^{(i)} \ V_{j2}^{(i)}), \quad i = 1, 2$$

be the ULVD of A_j computed by Algorithms 4.1 and 5.1, respectively.

Note that here we are discarding $U_j^{(i)}$, $i = 1, 2$. Finally, let

$$C_j^{(i)} = \bar{Y}_j^{(i)} \bar{\Sigma}_j^{(i)} \{\bar{W}_j^{(i)}\}^T, \quad \bar{W}_j^{(i)} = (\bar{W}_{j1}^{(i)} \ \bar{W}_{j2}^{(i)}), \quad i = 1, 2$$

by the SVD of $C_j^{(i)}$ computed by the one-sided Jacobi method. Define $\widetilde{W}_j$ by

$$\widetilde{W}_j^{(i)} = (\ \widetilde{W}_{j1}^{(i)} \ \ \widetilde{W}_{j2}^{(i)} \ ) = V_j^{(i)} \bar{W}_j^{(i)}, \quad i = 1, 2.$$

Define the angles between the subspaces

$$\sin \theta_1^{(i)} = \| \{W_{j1}^{(i)}\}^T V_{j2}^{(i)} \|, \quad \sin \theta_2^{(i)} = \| \{\widetilde{W}_{j1}^{(i)}\}^T V_{j2}^{(i)} \|, \quad i = 1, 2. \tag{5.38}$$

The angles θ_l, $l = 1, 2$ represent, respectively, error between the true noise subspace from the Jacobi SVD and the approximate one from tracking the ULVD, the approximation error from the ULV decomposition, and the subspace errors from ULV subspace tracking.

We plotted $\log_{10}(\sin \theta_1^{(1)})$ in dashed-dot, $\log_{10}(\sin \theta_1^{(2)})$ in solid, and $\log_{10}(\sin \theta_2^{(2)})$ in dotted line on the vertical axis of the second graph. $\sin \theta_2^{(2)}$ is the approximation errors discussed by Fierro and Hansen [43].

Finally, the TLS errors

$$\zeta_i = \frac{\|x^{(SVD)} - x_i^{(ULVD)}\|}{\|x_i^{(SVD)}\|}, \quad i = 1, 2$$

are given in logarithm in the last plot. Here, $x^{(SVD)}$ and $x_i^{(ULVD)}$, $i = 1, 2$ are the TLS solutions using the SVD and the ULVD with Algorithms 4.1 and 5.1, respectively. On the third graph of each figure we plotted ζ_1 in solid and ζ_2 in dashed-dot.

The following examples were also used in [13].

EXAMPLE 5.1. A, a 110-by-6 random matrix, b, a 110-by-1 random vector. Entries of A and b were chosen from a uniform distribution on the interval $(0, 1)$. 85 randomly chosen rows of $(A; b)$ were multiplied by $\gamma = 10^{-4}$ in order to vary the rank of the matrix, and $tol = 10^{-2}$. The window size p used was 12.

EXAMPLE 5.2. Same as Example 1 except that $\gamma = 10^{-8}$ and $tol = 10^{-6}$.

EXAMPLE 5.3. Same as Example 1 except that the matrix had an outlier of size 10^5 at $(18, 1)$ position.

The first plot shows that both algorithms estimated the numerical ranks correctly throughout the sliding window steps in spite of frequent rank changes. Thus the rank estimates from both algorithms and those by the Jacobi SVD were identical. As expected, the errors in the TLS solution are almost exactly the same as the size of $\sin \theta_1^{(i)}$, $i = 1, 2$ in (5.38).

The second plot in each figure shows that the noise subspace error is very small giving accurate TLS solutions. The quantities in (5.38) are shown to be essentially identical indicating that the subspace errors from our algorithm are from the rounding errors, not approximation errors.

Moreover, both algorithms perform well even when G becomes singular (indicated by '*' for Algorithm 4.1 and '‸' for Algorithm 5.1 in the first plot). We tested several other examples, and these results were typical.

As in [13] we used the Corrected Seminormal Equation (CSNE) technique whenever the downdating is not possible, namely, $\|a\| > 1$ in (4.5), and $\beta_1^2 + \beta_2^2 > 1$ in (5.6). We indicated the time steps where the CSNE was used with '+' for Algorithm 4.1 and '#' for Algorithm 5.1.

For Example 1, we plotted the norm estimates by Algorithm 5.1: $\|L^{-1}\|$ computed by the SVD (in solid), $\bar{\kappa}$ of Theorem 5.1 (in solid dot), and $\|L^{-1}\|_F$ computed by the Algorithm 5.1 as described in section 5.3.2 (dotted), again on a log scale, relative to *tol*. This verifies the bound from Theorem 5.1.
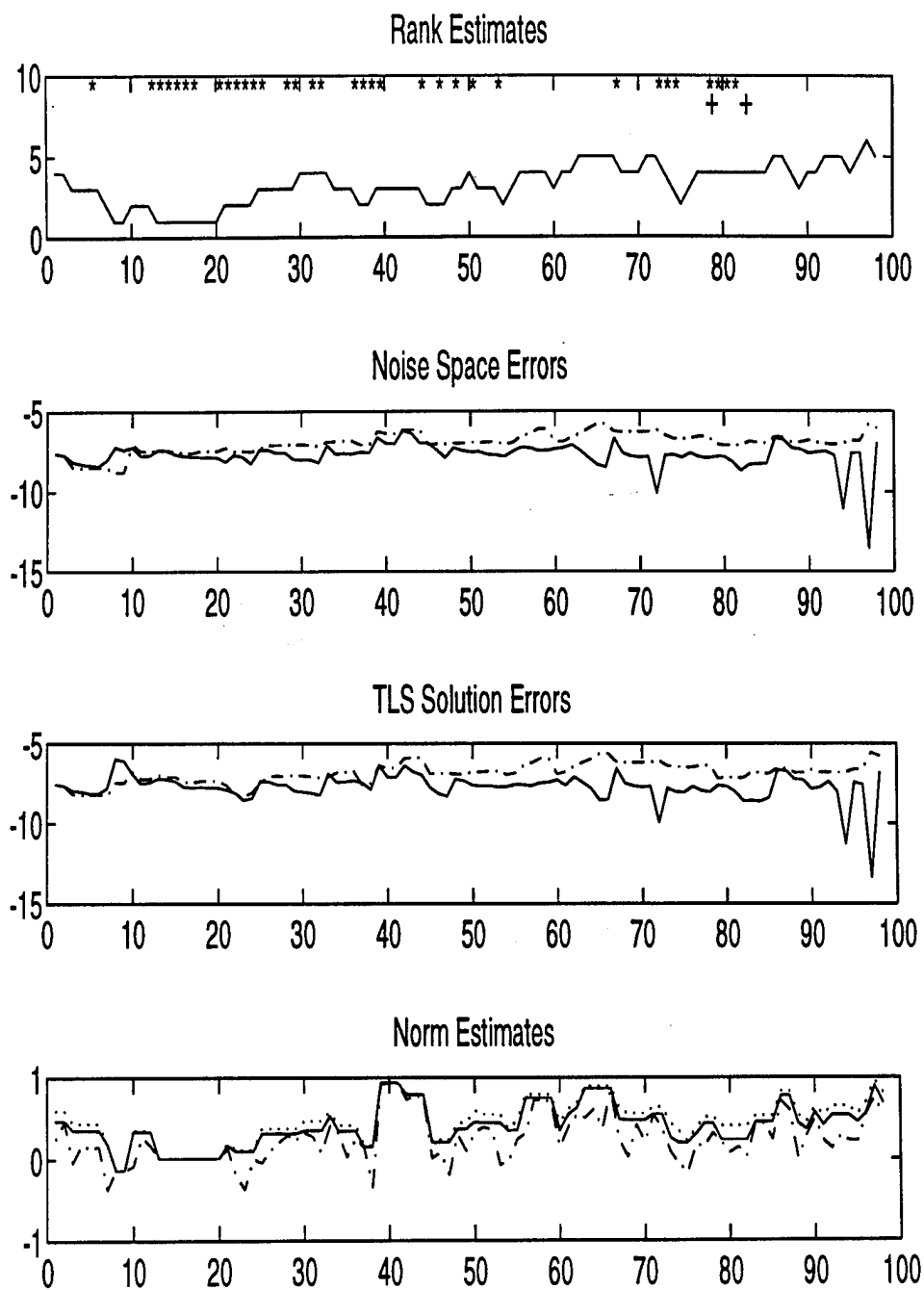
Fig. 5.3.   Example 5.1
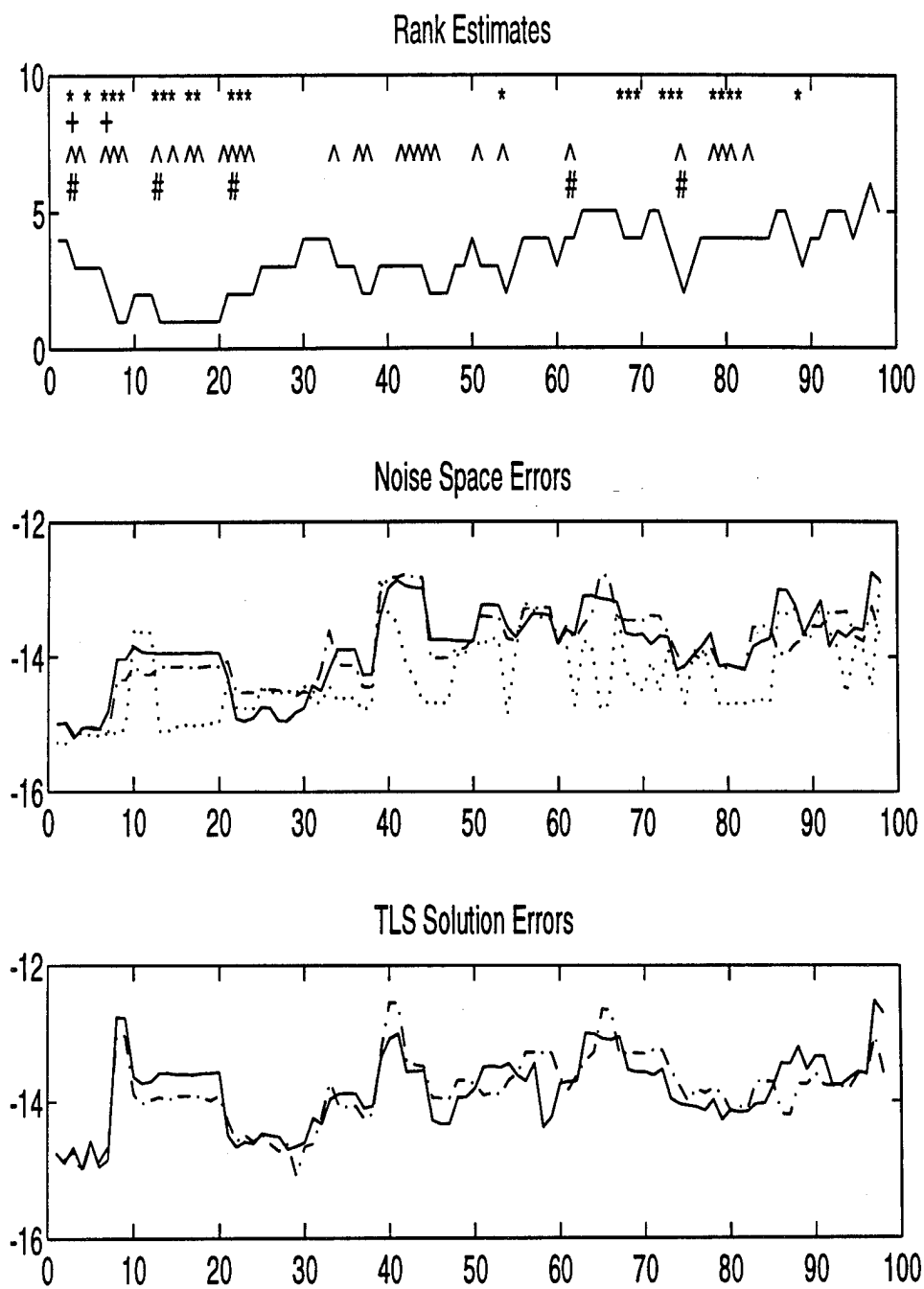
Fig. 5.4.   Example 5.2
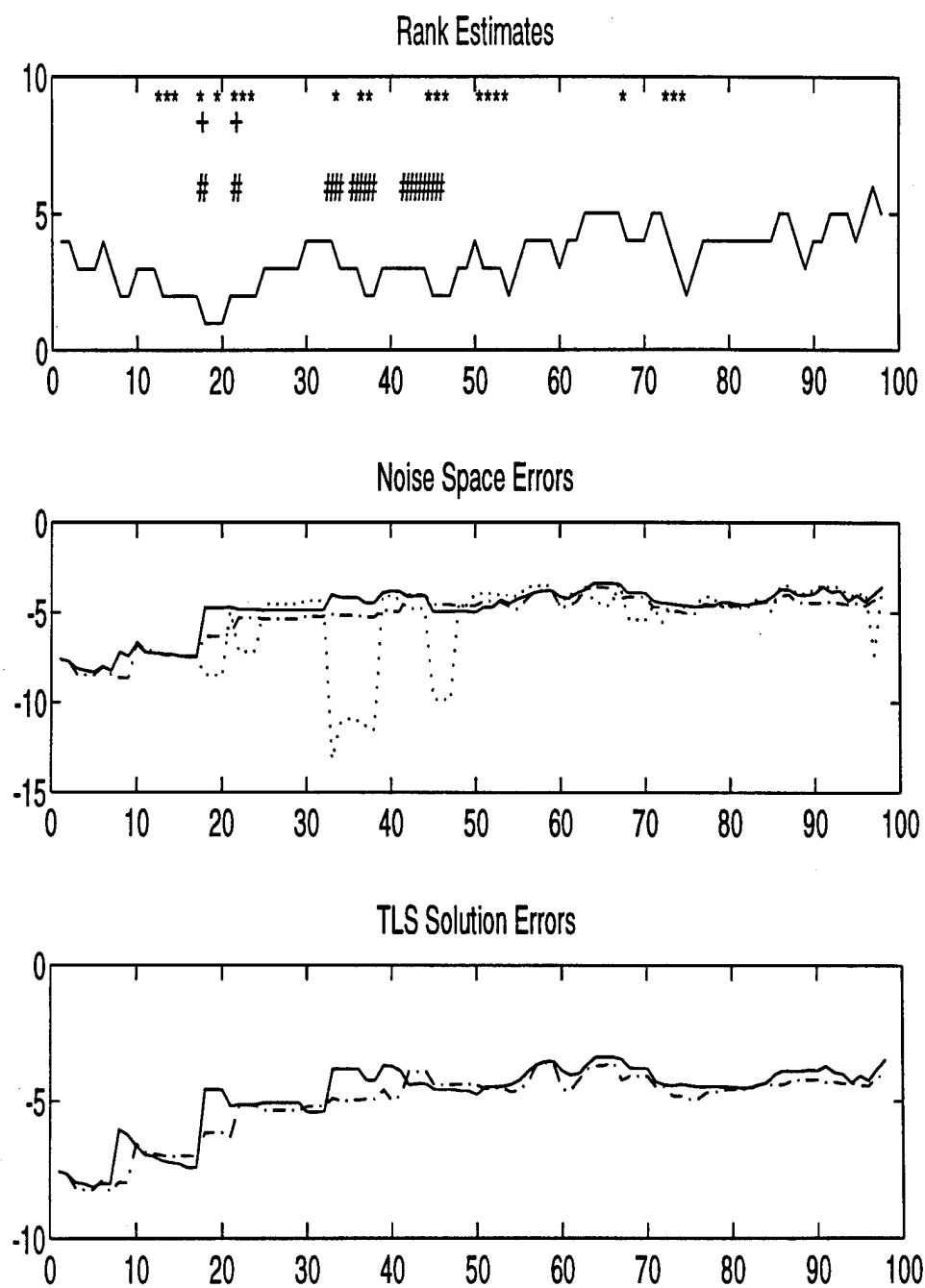
Fig. 5.5.   Example 5.3

Chapter 6

Modifying the Singular Value Decomposition

6.1 Introduction

We discuss methods for updating and downdating the SVD and partial SVD of $A \in \mathcal{R}^{m \times n}$ of the form (1.2) and (1.3). Throughout this chapter we use B in place of M in (1.1) to denote diagonal form or partially reduced bidiagonal form.

Unlike the Jacobi-type SVD updating procedures [79, 80, 81], we transform updating/downdating problem into a problem of finding a bidiagonal matrix $\bar{B}$ and an orthogonal matrix $\bar{V}$ such that

$$B^T B \pm z z^T = \bar{V} \bar{B}^T \bar{B} \bar{V}^T \tag{6.1}$$

where z is defined in 1.9). Throughout this chapter the bidiagonal matrix $\bar{B}$ has the form

$$\bar{B} = \begin{pmatrix} \gamma_1 & \phi_1 & 0 & \cdot & \cdot & 0 & 0 \\ 0 & \gamma_2 & \phi_2 & \cdot & \cdot & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \gamma_{n-2} & \phi_{n-2} & 0 \\ 0 & \cdot & \cdot & \cdot & \cdot & \gamma_{n-1} & \phi_{n-1} \\ 0 & 0 & \cdot & \cdot & \cdot & 0 & \gamma_n \end{pmatrix}.$$

We may also use the MATLAB-like shorthand

$$\bar{B} = \text{bidiag}(\gamma(1{:}n), \phi(1{:}n-1))$$

to denote the above bidiagonal matrix. As in modifying the ULVD from the previous chapters, our approaches to downdating the decompositions use ideas from chasing algorithms [1, 93, 115, 125] and from the downdating algorithm due to Saunders [46, 85].

The following are the main results of this chapter:

- Procedures for updating and downdating the SVD which obtain bidiagonal forms such that

$$\bar{B} = \begin{pmatrix} \bar{B}_1 & \phi_l e_l e_1^T \\ 0 & \bar{B}_2 \end{pmatrix} \begin{matrix} l \\ n-l \end{matrix}, \quad \left\| \begin{pmatrix} \phi_l e_l e_1^T \\ \bar{B}_2 \end{pmatrix} \right\| \leq \sigma_{k+1}(A) \qquad (6.2)$$

where $\bar{B}_1$ and $\bar{B}_2$ are upper bidiagonal, and $l = k+1$ for updating and $l = k$ for downdating. This form preserves more of the accuracy of the small singular values and is not achieved by standard chasing procedures. We can then use one of several algorithms to find the singular values of the bidiagonal matrix $\bar{B}$ to relative accuracy [11, 35, 40]. The singular vector matrix can be modified by a procedure due to Gu and Eisenstat[54] in $\mathcal{O}(mn)$ operations (the constant on mn depends upon machine precision). That is the same order of complexity as for the ULVD methods with similar stability properties.

- A perturbation theory for the singular subspaces from modified matrices and block-wise error bounds for the above procedures.

The condition (6.2) is achieved because the algorithms for both updating and downdating problems produce an orthogonal matrix $\bar{V}$ that has the form

$$
\bar{V} = \begin{pmatrix} \overset{k}{\bar{V}_{11}} & \overset{n-k}{0} \\ 0 & \bar{V}_{22} \end{pmatrix} \begin{matrix} k \\ n-k \end{matrix} .
\tag{6.3}
$$

There is never a rotation of the first k columns of B with the last $n - k$.

In the following section we describe the secular equation approach as an alternative to chasing algorithm. We show that this approach can fail to separate the singular values in separate blocks. In Section 6.3 we give some basic chasing procedures for modifying the SVD, and our chasing procedures which have the property (6.2). Section 6.4 gives the perturbation theory and discusses error analysis. Section 6.5 gives some computational examples.

6.2   Secular Equation Approach

The alternative to chasing algorithms for modifying the SVD is that of finding the zeroes of a particular spectral function [10, 25, 49, 53, 54, 67, 97],

$$
f(\bar{\sigma}) = 1 + \alpha \sum_{i=1}^{n} \frac{z_i^2}{\sigma_i^2 - \bar{\sigma}^2} = 0
\tag{6.4}
$$

where $\alpha > 0$ for updating and $\alpha < 0$ for downdating, and $\bar{\sigma}$ is the singular value of the modified matrix. The corresponding singular vector is given by

$$
\bar{v} = \frac{(B^T B - \bar{\sigma}^2 I_n)^{-1} z}{\|(B^T B - \bar{\sigma}^2 I_n)^{-1} z\|}.
\tag{6.5}
$$

Here, we assume that B is diagonal. That approach, as yet, does not allow us to separate the singular values into separate blocks as shown in the following example.

EXAMPLE 6.1. Let

$$B = \begin{pmatrix} 1 & 0 \\ 0 & 1 \cdot 10^{-10} \\ 1 & 1 \cdot 10^{-10} \end{pmatrix}.$$

QR decomposition B is given by $B = QR$ where

$$Q = \begin{pmatrix} -7.0711 \cdot 10^{-1} & 4.0825 \cdot 10^{-1} & -5.7735 \cdot 10^{-1} \\ 0 & -8.1650 \cdot 10^{-1} & -5.7735 \cdot 10^{-1} \\ -7.0711 \cdot 10^{-1} & -4.0825 \cdot 10^{-1} & 5.7735 \cdot 10^{-1} \end{pmatrix}$$

$$R = \begin{pmatrix} -1.4142 & -7.0711 \cdot 10^{-11} \\ 0 & -1.2247 \cdot 10^{-10} \\ 0 & 0 \end{pmatrix}.$$

The SVD of R is given by $R = USV^T$ where

$$U = \begin{pmatrix} -1.0000 & 4.3301 \cdot 10^{-21} & 0 \\ -4.3301 \cdot 10^{-21} & -1.0000 & 0 \\ 0 & 0 & 1.0000 \end{pmatrix},$$

$$S = \begin{pmatrix} 1.4142 & 0 \\ 0 & 1.2247 \cdot 10^{-10} \\ 0 & 0 \end{pmatrix}, \quad V = \begin{pmatrix} 1 & -5.0000 \cdot 10^{-11} \\ 5.0000 \cdot 10^{-11} & 1 \end{pmatrix}.$$

Let

$$\widehat{U} = QU = \begin{pmatrix} 7.0711 \cdot 10^{-1} & -4.0825 \cdot 10^{-1} & -5.7735 \cdot 10^{-1} \\ 3.5355 \cdot 10^{-21} & 8.1650 \cdot 10^{-1} & -5.7735 \cdot 10^{-1} \\ 7.0711 \cdot 10^{-1} & 4.0825 \cdot 10^{-1} & 5.7735 \cdot 10^{-1} \end{pmatrix}.$$

It can be verified that $B = \widehat{U} S V^T$. Let

$$\widehat{D} = B^T B = \begin{pmatrix} 1.0000 & 0 & 1.0000 \\ 0 & 1.0000 \cdot 10^{-20} & 1.0000 \cdot 10^{-20} \\ 1.0000 & 1.0000 \cdot 10^{-20} & 1.0000 \end{pmatrix}.$$

Then its engendecomposition is given by $\widehat{D} = X D X^T$ where

$$X = \begin{pmatrix} 0 & 7.0711 \cdot 10^{-1} & 7.0711 \cdot 10^{-1} \\ -1.0000 & 0 & 0 \\ 0 & 7.0711 \cdot 10^{-1} & -7.0711 \cdot 10^{-1} \end{pmatrix},$$

$$D = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 2.0000 & 0 \\ 0 & 0 & 2.2204 \cdot 10^{-16} \end{pmatrix}.$$

Notice that both the small eigenvalue of D (should be about $1 \cdot 10^{-20}$) and the subspaces are wrong!

6.3   Ordinary Chasing Algorithms

6.3.1   Basic Chasing Routines

In this section, we present an example of an orthogonal chasing scheme that produces orthogonal matrices $\bar{U}, \bar{V} \in \mathcal{R}^{n \times n}$ such that

$$\widehat{B} = \bar{U}^T B \bar{V}, \quad \bar{V}^T z = \rho e_1, \quad \rho = \|z\| \tag{6.6}$$

where $\widehat{B}$ is lower bidiagonal. For the 4×4 case, it is given in Fig. 6.1. Here r and x denote possibly large elements and e and y denote small elements. See Algorithm 3.4 for the formal description.

In theory, this could be used to produce an updated or downdated bidiagonal matrix very easily. We have

$$\bar{V}^T (B^T B \pm z z^T) \bar{V} = \widehat{B}^T \widehat{B} \pm \rho^2 e_1 e_1^T = \bar{B}^T \bar{B}$$

If $\widehat{B}^T = \text{bidiag}(\widehat{\gamma}(1:n), \widehat{\phi}(1:n-1))$, then $\bar{B}^T = \text{bidiag}(\bar{\gamma}(1:n), \bar{\phi}(1:n-1))$ is identical to $\widehat{B}^T$ except that $\bar{\gamma}_1 = \sqrt{\widehat{\gamma}_1^2 - \rho^2}$. This is illustrated in the last rotation in Fig. 6.1, denoted by a pair of $\overset{*}{\to}$. For updating it is simply a Givens rotation. It should be noted for downdating that the assumption $|\widehat{\gamma}_1| \geq \rho$ is equivalent to the assumption that $B^T B - z z^T$ is positive semi-definite.

Such a procedure shown in Fig. 6.1 does not preserve the separation of subspaces for large and small singular values as accurately as we would like. Large elements can get chased down into the lower part of the bidiagonal matrix $\widehat{B}$ as shown in the following example.

$$
\begin{array}{cc}
\downarrow\ \downarrow & \\
\end{array}
$$

$$
\begin{pmatrix}
r & & & \\
 & r & & \\
 & & e & \\
 & & & e \\
x & x & y & \hat{y}
\end{pmatrix}
\begin{matrix}\rightarrow\\\rightarrow\end{matrix}
\begin{pmatrix}
r & & & \\
 & r & & \\
 & & e & \hat{e} \\
 & & e & e \\
x & x & y & 0
\end{pmatrix}
\begin{pmatrix}
r & & & \\
 & r & & \\
 & & e & \\
 & & e & e \\
x & x & \hat{y} & 0
\end{pmatrix}
\begin{matrix}\rightarrow\\\rightarrow\end{matrix}
\begin{pmatrix}
r & & & \\
 & r & \hat{r} & \\
 & e & e & \\
 & e & e & e \\
x & x & 0 & 0
\end{pmatrix}
$$

$$
\begin{matrix}\rightarrow\\\rightarrow\end{matrix}
\begin{pmatrix}
r & & & \\
 & r & & \\
 & r & r & \\
 & \hat{e} & e & e \\
x & x & 0 & 0
\end{pmatrix}
\begin{pmatrix}
r & & & \\
 & r & & \\
 & r & r & \hat{e} \\
 & & e & e \\
x & x & 0 & 0
\end{pmatrix}
\begin{pmatrix}
r & & & \\
 & r & & \\
 & r & r & \\
 & & e & e \\
x & \hat{x} & 0 & 0
\end{pmatrix}
\begin{matrix}\rightarrow\\\rightarrow\end{matrix}
\begin{pmatrix}
r & \hat{r} & & \\
r & r & & \\
r & r & r & \\
 & & e & e \\
\rho & 0 & 0 & 0
\end{pmatrix}
$$

$$
\begin{matrix}\rightarrow\\\rightarrow\end{matrix}
\begin{pmatrix}
r & & & \\
r & r & & \\
\hat{r} & r & r & \\
 & & e & e \\
\rho & 0 & 0 & 0
\end{pmatrix}
\begin{pmatrix}
r & & & \\
r & r & \hat{r} & \\
 & r & r & \\
 & & e & e \\
\rho & 0 & 0 & 0
\end{pmatrix}
\begin{matrix}\rightarrow\\\rightarrow\end{matrix}
\begin{pmatrix}
r & & & \\
r & r & & \\
 & r & r & \\
 & \hat{e} & e & e \\
\rho & 0 & 0 & 0
\end{pmatrix}
\begin{pmatrix}
r & & & \\
r & r & & \\
 & r & r & \hat{r} \\
 & & r & e \\
\rho & 0 & 0 & 0
\end{pmatrix}
$$

$$
\begin{matrix}\bullet\\\rightarrow\\\\\bullet\\\rightarrow\end{matrix}
\begin{pmatrix}
r & & & \\
r & r & & \\
 & r & r & \\
 & & r & r \\
\hat{\rho} & 0 & 0 & 0
\end{pmatrix}
\begin{pmatrix}
\tilde{r} & & & \\
r & r & & \\
 & r & r & \\
 & & r & r \\
0 & 0 & 0 & 0
\end{pmatrix}
$$

Fig. 6.1.   Ordinary Chasing Procedure

EXAMPLE 6.2. Suppose we have

$$B = \mathrm{diag}(0.2071, 1.510 \cdot 10^{-3}, 8.081 \cdot 10^{-4}, 6.383 \cdot 10^{-4}, 5.184 \cdot 10^{-7})$$

$$z = (7.964 \cdot 10^{-3}, 8.012 \cdot 10^{-3}, -9.102 \cdot 10^{-3}, -2.821 \cdot 10^{-3}, 1.607 \cdot 10^{-2})^T$$

After **forchase** is applied to

$$\begin{pmatrix} B \\ z^T \end{pmatrix}, \tag{6.7}$$

we obtain $\widehat{B}^T = \mathrm{bidiag}(\gamma(1\!:\!5), \phi(1\!:\!4))$ where

$$\gamma(1\!:\!5) = (7.864 \cdot 10^{-2}, -5.357 \cdot 10^{-2}, -1.317 \cdot 10^{-3}, -7.283 \cdot 10^{-4}, -6.414 \cdot 10^{-4})^T$$

$$\phi(1\!:\!4) = (-0.1852, 4.134 \cdot 10^{-5}, -4.030 \cdot 10^{-4}, 8.115 \cdot 10^{-5})^T$$

Here $\gamma(5) > tol$, so that the smallest singular values may be overestimated. In fact, there should be no rank increase as we will see in Example 6.3.

In the next section, we show that **forchase** and **backchase** procedures described in Section 3.2.1 can be combined in a fashion that allow us to update or downdate the SVD more accurately.

6.3.2 The Updating Algorithm

Let

$$B = \mathrm{diag}(\gamma(1\!:\!n)), \quad \gamma_1 \geq \cdots \geq \gamma_k > \epsilon > \gamma_{k+1} \geq \cdots \geq \gamma_n.$$

We partition B into

$$B_1 = \text{diag}(\gamma(1\!:\!k)), \quad B_2 = \text{diag}(\gamma(k+1\!:\!n)).$$

Let z be defined in (1.9). Then the following algorithm updates the diagonal matrix into upper bidiagonal matrix.

ALGORITHM 6.1 (PROCEDURE FOR UPDATING DIAGONAL MATRIX). Given input $\gamma(1\!:\!n)$ that contains $\sigma_i(A), i = 1, \ldots, n$ and the update vector z, this procedure produces the updated bidiagonal matrix $B = \text{bidiag}(\gamma(1\!:\!n), \phi(1\!:\!n-1))$. We also input k the number of singular values greater than *tol*.

Step 1. Compute orthogonal matrices $\bar{U}_1, \bar{V}_1 \in \mathcal{R}^{k \times k}$ and $\bar{U}_2, \bar{V}_2 \in \mathcal{R}^{(n-k) \times (n-k)}$ such that

$$\bar{U}_1^T B_1 \bar{V} = B_1^{(1)}, \quad \bar{V}_1 x = \rho_1 e_k^{(k)}, \quad \rho_1 = \|x\| \tag{6.8}$$

$$\bar{U}_2^T B_2 \bar{V}_2 = B_2^{(1)}, \quad \bar{V}_2 y = \rho_2 e_1^{(n-k)}, \quad \rho_2 = \|y\| \tag{6.9}$$

where $B_1^{(1)}$ is upper bidiagonal, and $B_2^{(1)}$ lower bidiagonal.

Step 2. Let $Q_1 = J(1, 3, \theta_1)$ and $Q_2 = J(2, 3, \theta_2)$ define Givens rotations for some $\theta_i, i = 1, 2$ such that

$$Q_2^T Q_1^T \begin{pmatrix} \rho_1 & \rho_2 \\ \gamma_k^{(1)} & 0 \\ 0 & \gamma_{k+1}^{(1)} \end{pmatrix} = \begin{pmatrix} \gamma_k^{(2)} & \phi_k^{(2)} \\ 0 & \gamma_{k+1}^{(2)} \\ 0 & 0 \end{pmatrix},$$

that is, use Algorithm 3.8 for updating 2×2 matrix,

$$[\gamma_k^{(2)}, \phi_k^{(2)}, \gamma_{k+1}^{(2)}] = \mathbf{up22}(\gamma_k^{(1)}, 0, \gamma_{k+1}^{(1)}, \rho_1, \rho_2).$$

Thus if we let $\bar{U}_3 = J(k, n+1, \theta_1) \, J(k+1, n+1, \theta_2)$, then we have

$$
\begin{pmatrix}
B_1^{(2)} & \phi_k^{(2)} e_k e_1^T \\
0 & B_2^{(2)} \\
0 & 0
\end{pmatrix}
= \bar{U}_3^T
\begin{pmatrix}
B_1^{(1)} & 0 \\
0 & B_2^{(2)} \\
\rho_1 e_k^T & \rho_2 e_1^T
\end{pmatrix}.
$$

Step 3. Use Algorithm 3.6 to construct an orthogonal matrix $\bar{U}_4 \in \mathcal{R}^{(n-k)\times(n-k)}$

such that

$$
\bar{B} =
\begin{pmatrix}
\bar{B}_1 & \bar{\phi}_k \\
0 & \bar{B}_2
\end{pmatrix}
=
\begin{pmatrix}
I_k & 0 \\
0 & \bar{U}_4
\end{pmatrix}
\begin{pmatrix}
B_1^{(2)} & \phi_k^{(2)} e_k e_1^T \\
0 & B_2^{(2)}
\end{pmatrix}
$$

where $\bar{B}_2$ is upper bidiagonal. Thus $\bar{U}$ and $\bar{V}$ are given by

$$
\bar{U} =
\begin{pmatrix}
1 & 0 & 0 \\
0 & \bar{U}_1 & 0 \\
0 & 0 & \bar{U}_2
\end{pmatrix}
\bar{U}_3
\begin{pmatrix}
I_k & 0 & 0 \\
0 & \bar{U}_4 & 0 \\
0 & 0 & 1
\end{pmatrix}, \quad
\bar{V} =
\begin{pmatrix}
\bar{V}_1 & 0 \\
0 & \bar{V}_2
\end{pmatrix}. \tag{6.10}
$$

Note that after Step 1 the updating problem is reduced to a 2×2 problem. Step

3 is to restore the matrix to upper bidiagonal form, which is equivalent to a step of **qd**

procedure. The block diagonal form of $\bar{V}$ in (6.10) is highly significant. Any modification

of the subspaces associated with the first k singular values and the last $n - k$ singular values will be computed in the reduction of the bidiagonal matrix $\bar{B}$.

Fig. 6.2–6.3 show the reduction steps for this algorithm with $n = 7$ and $k = 4$. Here, a pair of $\overset{*}{\to}$ denotes the application of Givens rotations that corresponds to Step 2. Unlike the ordinary chasing algorithm, Algorithm 6.1 preserves the block structure as illustrated in the following example.

EXAMPLE 6.3. When Algorithm 6.1 is applied to (6.7), we obtain

$$\gamma(1{:}5) = (6.441 \cdot 10^{-4}, 9.869 \cdot 10^{-4}, 3.974 \cdot 10^{-3}, 5.009 \cdot 10^{-2}, 3.818 \cdot 10^{-7})^T$$

$$\phi(1{:}4) = (1.054 \cdot 10^{-4}, 7.309 \cdot 10^{-4}, 0.1862, -4.444 \cdot 10^{-8})^T$$

keeping the separation of the large and small blocks, and so preserving the accuracy of the tiny singular values.

6.3.3 The Downdating Algorithm

An immediate difference between the updating and downdating procedures is that we write B in the form

$$B = \text{diag}(B_1, B_2, O_{n-p}) \tag{6.11}$$

where

$$B_1 = \text{diag}(\gamma(1{:}k)) = \text{diag}(\sigma_1, \ldots, \sigma_k),$$
$$B_2 = \text{diag}(\gamma(k+1{:}p)) = \text{diag}(\sigma_{k+1}, \ldots, \sigma_p),$$

and

$$\sigma_k > \epsilon > \sigma_{k+1}, \quad \sigma_{p+1} = \cdots = \sigma_n = 0,$$

Fig. 6.2.   Bidiagonal Reduction Steps for Modifying the SVD

$$
\xrightarrow{\;\cdot\;}\xrightarrow{\;\cdot\;}
\begin{pmatrix}
r & r & & & & & & \\
  & r & r & & & & & \\
  &   & r & r & & & & \\
  &   &   & r & & & & \\
  &   &   &   & & e & & \\
  &   &   &   & & e & e & \\
  &   &   &   & &   & e & e \\
  &   &   & \widehat{x} & y & & &
\end{pmatrix}
\xrightarrow{\;\cdot\;}\xrightarrow{\;\cdot\;}
\begin{pmatrix}
r & r & & & & & & \\
  & r & r & & & & & \\
  &   & r & r & & & & \\
  &   &   & r & y & & & \\
  &   &   &   & e & & & \\
  &   &   &   & e & e & & \\
  &   &   &   &   & e & e & \\
  &   &   &   & \widehat{y} & & &
\end{pmatrix}
\xrightarrow{\;}\xrightarrow{\;}
\begin{pmatrix}
r & r & & & & & & \\
  & r & r & & & & & \\
  &   & r & r & & & & \\
  &   &   & r & r & & & \\
  &   &   &   & & e & & \\
  &   &   &   & & \widehat{e} & e & \\
  &   &   &   & &   & e & e
\end{pmatrix}
$$

$$
\xrightarrow{\;}\xrightarrow{\;}
\begin{pmatrix}
r & r & & & & & \\
  & r & r & & & & \\
  &   & r & r & & & \\
  &   &   & r & r & & \\
  &   &   &   & e & e & \\
  &   &   &   & & e & \\
  &   &   &   & & \widehat{e} & e
\end{pmatrix}
\begin{pmatrix}
r & r & & & & & \\
  & r & r & & & & \\
  &   & r & r & & & \\
  &   &   & r & r & & \\
  &   &   &   & e & e & \\
  &   &   &   & & e & e \\
  &   &   &   & & & e
\end{pmatrix}
$$

Fig. 6.3. 2×2 Updating/Downdating Steps and a qd Step

thus allowing for the possibility that some singular values are exactly zero. As in downdating the ULVD, we partition the vector z in the form,

$$z = V^T r = \begin{pmatrix} x \\ y \\ y_0 \end{pmatrix} \begin{matrix} k \\ p-k \\ n-p \end{matrix} \quad .$$

Here, y_0 is presumed to be the result of rounding errors and is ignored. However, if $\|y_0\| \gg \mu * \sigma_1(A)$, then we should not downdate, we should refactor.

Even when there are no zero singular values, and even though $z = V^T r$ and r is the first row of A in (1.1), $B^T B - xx^T$ is positive semi-definite. However, occasionally, even that is not the case. The usual way to test if $B^T B - xx^T$ is positive semi-definite is to solve

$$B_1^T s = x. \tag{6.12}$$

If $\|s\| > 1$, then we cannot downdate B_1 by x. One possible remedy is to try to obtain a better value for $x = V_1^T A^T e_1$. That can be done using the corrected semi-normal equations (CSNE) [18, 20] as we have used for modifying the ULVD in Chapter 4.

If $\|s\| > \eta$ where $\eta < 1$, then solve

$$B_1 c = s. \tag{6.13}$$

It should be noted that c solves the least squares problem

$$\min_{c \in \mathcal{R}^k} \|AV_1 c - e_1\|$$

and its value can be improved by the iterative improvement steps

$$r = e_1 - AV_1 c \tag{6.14}$$

$$\delta x = V_1^T A^T r \tag{6.15}$$

$$B_1^T \delta s = \delta x \tag{6.16}$$

$$s \leftarrow s + \delta s, \quad x \leftarrow x + \delta x \tag{6.17}$$

At this point, if $\|s\| > 1$, we signal that downdating is not possible, and thus other options should be considered, such as refactoring or choosing a higher threshold ϵ. Otherwise, the algorithm proceeds in a similar manner to Algorithm 6.1.

We now present the downdating algorithm.

ALGORITHM 6.2 (PROCEDURE FOR DOWNDATING DIAGONAL MATRIX). Given the input $\gamma(1:n)$ that contains $\sigma_i(A), i = 1, \ldots, n$ and the update vector z of the form (4.3), this procedure produces the downdated bidiagonal matrix $B = \text{bidiag}(\gamma(1:n), \phi(1:n-1))$. We also input k the number of singular values greater than tol. y_0 is ignored unless $\|y_0\| \gg \mu$.

Step 1. If $\|s\| \geq \eta$ where s is defined in (6.12) and $\eta < 1$, then solve (6.13)–(6.16) for δs and δx. Update s and x as in (6.17). If $\|s\| > 1$ or $\|y_0\| > tol \approx 100 * \mu$, then quit and exit; otherwise, do Steps 2–4.

Step 2. Same as Step 1 of Algorithm 6.1.

Step 3. Compute

$$a_1 = \rho/\gamma_k^{(1)}, \quad a_2 = \min\left\{ \rho_2/\gamma_{k+1}^{(1)}, \sqrt{1 - a_1^2} \right\}, \tag{6.18}$$

$$\alpha = \sqrt{1 - a_1^2 - a_2^2} \tag{6.19}$$

We can then find two 3×3 Givens rotations $Q_i = J(1, i+1, \theta_i)$, $i = 1, 2$ such that

$$Q_1 Q_2 \begin{pmatrix} \alpha \\ a_1 \\ a_2 \end{pmatrix} = e_1 .$$

In that case we modify the k-th and $(k+1)$-st rows of the matrix

$$\begin{pmatrix} \gamma_k^{(2)} & \phi_k^{(2)} \\ 0 & \gamma_{k+1}^{(2)} \\ \rho_1 & \tilde{\rho}_2 \end{pmatrix} = Q_1 Q_2 \begin{pmatrix} \gamma_k^{(1)} & 0 \\ 0 & \gamma_{k+1}^{(1)} \\ 0 & 0 \end{pmatrix},$$

where $\tilde{\rho}_2 = \gamma_{k+1}^{(1)} \sqrt{1 - \alpha^2}$. This can be done by using Algorithm 3.9,

$$[\gamma_k^{(2)}, \phi_k^{(2)}, \gamma_{k+1}^{(2)}] = \mathbf{down22}(\gamma_k^{(1)}, 0, \gamma_{k+1}^{(1)}, a_1, a_2).$$

Thus if we let $\bar{U}_3 = J(1, k+2, \theta_2)^T J(1, k+1, \theta_1)^T$, then we have

$$\begin{pmatrix} B_1^{(2)} & \gamma_k^{(2)} e_k e_1^T \\ 0 & B_2^{(2)} \\ \rho_1 e_k^T & \rho_2 e_1^T \end{pmatrix} = \bar{U}_3^T \begin{pmatrix} B_1^{(1)} & 0 \\ 0 & B_2^{(2)} \\ 0 & 0 \end{pmatrix} .$$

Step 4. Same as Step 3 of Algorithm 6.1.

We note that in (6.18) $|a_1| = \|B_1^{-1} x\| = \|s\|$. Thus B_1 can be downdated by x if and only if $|a_1| \le 1$. For Algorithm 6.2, we assume that is the case. If $\alpha = 0$, but $a_2 \ne 0$,

then $\gamma_{k+1}^{(2)} = 0$, whereas if $\alpha = a_2 = 0$, then $\gamma_k^{(2)} = \phi_k^{(2)} = 0$. Fig. 6.2–6.3 show the reduction steps for this algorithm with $n = 7$ and $k = 4$, but, this time, a pair of $\overset{*}{\to}$ denotes the application of rotations of Saunders' algorithm that corresponds to Step 3.

Thus we have simple algorithms to perform either an update or downdate. The downdating procedure has the following consistency property similar to Proposition 4.2.

PROPOSITION 6.1. *Assume that Algorithm 6.2 is done in exact arithmetic, that U and V in (4.1) are exactly orthogonal, that $\widetilde{U} = U\bar{U}$ satisfies $\widetilde{U}e_1 = e_1$, and that $z = V^T r$ is computed exactly. Also let $\widetilde{V} = V\bar{V}$ and $\bar{z} = \widetilde{V}^T r = \rho_1 e_k + \rho_2 e_{k+1}$. If*

$$A + \delta A = \begin{pmatrix} r^T + \delta r^T \\ \bar{A} + \delta\bar{A} \end{pmatrix} = U \begin{pmatrix} B \\ 0 \end{pmatrix} V^T, \tag{6.20}$$

then

$$A + \delta A_0 = \begin{pmatrix} r^T \\ \bar{A} + \delta\bar{A} \end{pmatrix} = \widetilde{U} \begin{pmatrix} \bar{z}^T \\ \bar{B} \\ 0 \end{pmatrix} \widetilde{V}^T. \tag{6.21}$$

Thus $\|\delta A_0\| = \|\delta\bar{A}\| \le \|\delta A\|$.

Proof. We have that

$$\widetilde{U}^T A \widetilde{V} = \begin{pmatrix} \rho_1 e_k & \rho e_1 \\ \bar{B}_1 & \bar{\phi}_k e_k e_1^T \\ 0 & \bar{B}_2 \end{pmatrix} \tag{6.22}$$

and

$$\bar{U}^T \begin{pmatrix} B \\ 0 \end{pmatrix} \bar{V} = \begin{pmatrix} \rho_1 e_k & (\rho_2 + \delta\rho_2)e_1 \\ \bar{B}_1 & \phi_k e_k e_1^T \\ 0 & \bar{B}_2 \end{pmatrix}. \tag{6.23}$$

Thus from (6.20), $\|\delta r\|^2 = \delta\rho_1^2 + \delta\rho_2^2$. Using (6.23) and noting that $\tilde{U}e_1 = e_1$, we have

$$\tilde{U}^T \begin{pmatrix} \bar{z}^T \\ \bar{B} \end{pmatrix} \tilde{V} = \begin{pmatrix} r^T \\ \bar{A} + \delta\bar{A} \end{pmatrix}. \tag{6.24}$$

Thus comparing (6.24) with (6.21), we have $\delta A_0^T = \begin{pmatrix} 0 & \delta\bar{A}^T \end{pmatrix}$. The result immediately follows. $\square$

We note that Proposition 6.1 is merely a consistency property. What it says is that approximation used in Step 3 of Algorithm 6.2 does not increase the error over that caused by assuming that $\tilde{U}e_1 = e_1$.

6.3.4 Extensions to Partially Reduced Bidiagonal Forms

Algorithm 6.1 and 6.2 can be easily extended to the case where either B_1 or B_2 is bidiagonal as long as they are decoupled. We need only modify Step 1 of Algorithm 6.1 and Step 2 of Algorithm 6.2. Van Huffel and Park [115] describe chasing algorithms that given B_1 upper bidiagonal, produce orthogonal matrices $\bar{U}_1, \bar{V}_1 \in \mathcal{R}^{k \times k}$ such that

$$\bar{B}_1 = \bar{U}_1^T B_1 \bar{V}_1, \quad \bar{V}_1^T x = \rho_1 e_k$$

where $\bar{B}_1$ is upper bidiagonal.

Fig. 6.4 shows how such an algorithm would work on a 5 × 5 example. Thus, using algorithms such as the zero-shift QR [35] or the **qd** algorithm [40], it is possible to

$$
\begin{array}{c}\downarrow \ \downarrow \\ \begin{pmatrix} \hat{x} & x & x & x & x \\ r & r & & & \\ & r & r & & \\ & & r & r & \\ & & & r & r \\ & & & & r \end{pmatrix}\end{array}
\rightarrow
\begin{pmatrix} 0 & x & x & x & x \\ r & r & & & \\ \hat{r} & r & r & & \\ & & r & r & \\ & & & r & r \\ & & & & r \end{pmatrix}
\quad
\begin{array}{c}\downarrow \ \downarrow \\ \begin{pmatrix} 0 & \hat{x} & x & x & x \\ r & r & & & \\ & r & r & & \\ & & r & r & \\ & & & r & r \\ & & & & r \end{pmatrix}\end{array}
\begin{array}{c}\rightarrow \\ \rightarrow\end{array}
\begin{pmatrix} 0 & 0 & x & x & x \\ r & r & r & & \\ & r & r & & \\ \hat{r} & r & r & & \\ & & & r & r \\ & & & & r \end{pmatrix}
$$

$$
\begin{array}{c}\downarrow \ \downarrow \\ \begin{pmatrix} 0 & 0 & \hat{x} & x & x \\ r & r & r & & \\ & r & r & & \\ & & r & r & \\ & & & r & r \\ & & & & r \end{pmatrix}\end{array}
\begin{array}{c}\rightarrow \\ \rightarrow\end{array}
\begin{pmatrix} 0 & 0 & 0 & x & x \\ r & r & r & r & \\ & r & r & r & \\ & & r & r & \\ & & \hat{r} & r & r \\ & & & & r \end{pmatrix}
\begin{array}{c}\rightarrow \\ \rightarrow\end{array}
\begin{pmatrix} 0 & 0 & 0 & x & x \\ r & r & r & \hat{r} & \\ & r & r & r & \\ & & r & r & \\ & & & r & r \\ & & & & r \end{pmatrix}
\quad
\begin{array}{c}\downarrow \ \downarrow \\ \begin{pmatrix} 0 & 0 & 0 & x & x \\ r & r & r & & \\ \hat{r} & r & r & r & \\ & & r & r & \\ & & & r & r \\ & & & & r \end{pmatrix}\end{array}
$$

$$
\begin{array}{c}\downarrow \ \downarrow \\ \begin{pmatrix} 0 & 0 & 0 & \hat{x} & x \\ r & r & r & & \\ & r & r & r & \\ & & r & r & \\ & & & r & r \\ & & & & r \end{pmatrix}\end{array}
\begin{array}{c}\rightarrow \\ \rightarrow\end{array}
\begin{pmatrix} 0 & 0 & 0 & 0 & x \\ r & r & r & & \\ & r & r & r & r \\ & & r & r & r \\ & & & r & r \\ & & & \hat{r} & r \end{pmatrix}
\rightarrow
\begin{pmatrix} 0 & 0 & 0 & 0 & x \\ r & r & r & & \\ & r & r & r & \hat{r} \\ & & r & r & r \\ & & & r & r \\ & & & & r \end{pmatrix}
\quad
\begin{array}{c}\downarrow \ \downarrow \\ \begin{pmatrix} 0 & 0 & 0 & 0 & x \\ r & r & r & & \\ \hat{r} & r & r & r & \\ & & r & r & \\ & & & r & r \\ & & & & r \end{pmatrix}\end{array}
$$

$$
\begin{array}{c}\rightarrow \\ \rightarrow\end{array}
\begin{pmatrix} 0 & 0 & 0 & 0 & x \\ r & r & r & & \\ & r & r & r & \\ & & r & r & \hat{r} \\ & & & r & r \\ & & & & r \end{pmatrix}
\quad
\begin{array}{c}\downarrow \ \downarrow \\ \begin{pmatrix} 0 & 0 & 0 & 0 & x \\ r & r & r & & \\ & r & r & r & \\ & & r & r & \\ & & \hat{r} & r & r \\ & & & & r \end{pmatrix}\end{array}
\begin{array}{c}\rightarrow \\ \rightarrow\end{array}
\begin{pmatrix} 0 & 0 & 0 & 0 & x \\ r & r & r & r & \hat{r} \\ & r & r & r & \\ & & r & r & \\ & & & r & r \\ & & & & r \end{pmatrix}
\quad
\begin{array}{c}\downarrow \ \downarrow \\ \begin{pmatrix} 0 & 0 & 0 & 0 & x \\ r & r & r & & \\ \hat{r} & r & r & r & \\ & & r & r & \\ & & & r & r \\ & & & & r \end{pmatrix}\end{array}
$$

$$
\begin{array}{c}\rightarrow \\ \rightarrow\end{array}
\begin{pmatrix} 0 & 0 & 0 & 0 & x \\ r & r & r & & \\ & r & r & \hat{r} & \\ & & r & r & \\ & & & r & r \\ & & & & r \end{pmatrix}
\quad
\begin{array}{c}\downarrow \ \downarrow \\ \begin{pmatrix} 0 & 0 & 0 & 0 & x \\ r & r & r & & \\ & r & r & & \\ \hat{r} & r & r & & \\ & & r & r & \\ & & & r & r \end{pmatrix}\end{array}
\rightarrow
\begin{pmatrix} 0 & 0 & 0 & 0 & x \\ r & r & \hat{r} & & \\ & r & r & & \\ & & r & r & \\ & & & r & r \\ & & & & r \end{pmatrix}
\quad
\begin{array}{c}\downarrow \ \downarrow \\ \begin{pmatrix} 0 & 0 & 0 & 0 & x \\ r & r & & & \\ \hat{r} & r & r & & \\ & & r & r & \\ & & & r & r \\ & & & & r \end{pmatrix}\end{array}
$$

$$
\begin{pmatrix} 0 & 0 & 0 & 0 & x \\ r & r & & & \\ & r & r & & \\ & & r & r & \\ & & & r & r \\ & & & & r \end{pmatrix}
$$

Fig. 6.4. Reduction Steps for the Partially Reduced Bidiagonal Form

find the singular values that are below a certain threshold, and thus obtain a partially bidiagonal matrix of the form (1.3).

6.4 Error Analysis

6.4.1 Error Bounds for Blockwise Algorithms

We now present error bounds for the process of one update or downdate using the procedures in Sections 6.3.2 and 6.3.3. All of the matrices below are computed except those with δ in front of them.

The following two propositions are proven in the Appendix of [14].

PROPOSITION 6.2. *Algorithm 6.1 produces an updated matrix $\bar{B}$ such that for some orthogonal matrix $\bar{U}$, and $\bar{V}$ we have*

$$\bar{U}^T \begin{pmatrix} z^T \\ B \end{pmatrix} \bar{V} = \begin{pmatrix} 0 \\ \bar{B} + \delta\bar{B} + \delta\bar{B}_0 \end{pmatrix}$$

where

$$\delta\bar{B} = \text{diag}(\delta\bar{B}_1, \delta\bar{B}_2)$$

$$\delta\bar{B}_0 = \delta\gamma_k e_k e_k^T + \delta\phi_{k+1} e_k e_{k+1}^T + \delta\gamma_{k+1} e_{k+1} e_{k+1}^T$$

$$\|\delta\bar{B}_1\| \leq \mu f_1(n) \|B_1\| + \mathcal{O}(\mu^2)$$

$$\|\delta\bar{B}_2\| \leq \mu f_2(n) \|B_2\| + \mathcal{O}(\mu^2) = \mu f_2(n) \sigma_{k+1}(B) + \mathcal{O}(\mu^2)$$

$$|\delta\gamma_j| \leq \mu f_3(n) |\bar{\gamma}_j| + \mathcal{O}(\mu^2), \quad j = k, k+1$$

$$|\delta\phi_{k+1}| \leq \mu f_4(n) |\bar{\phi}_{k+1}| + \mathcal{O}(\mu^2)$$

where $f_i(n) = \mathcal{O}(n^2)$, $i = 1, 2$, and $f_i(n) = \mathcal{O}(n)$, $i = 3, 4$.

From [11, 35] this relative change in the entries of the bidiagonal matrix makes only relative changes in the singular values. Thus this update procedure is very stable.

PROPOSITION 6.3. *Algorithm 6.2 produces a downdated matrix $\bar{B}$ such that for some orthogonal matrices $\bar{U}$ and $\bar{V}$ we have*

$$\bar{U}^T \left(\begin{array}{c} B + \delta B \\ \\ 0 \end{array} \right) \bar{V} = \left(\begin{array}{c} (z + \delta z)^T \bar{V} \\ \\ \bar{B} \end{array} \right)$$

where

$$\delta B = \operatorname{diag}(\delta B_1, \delta B_2)$$

$$\|\delta B_1\| \le \mu f_5(n) \|B_1\| + \mathcal{O}(\mu^2)$$

$$\|\delta B_2\| \le \mu f_6(n) \|B_2\| + \mathcal{O}(\mu^2) = \mu f_6(n) \, \sigma_{k+1}(B) + \mathcal{O}(\mu^2)$$

$$\|\delta z(1{:}k)\| \le \mu f_7(n) \|z(1{:}k)\| + \mathcal{O}(\mu^2)$$

$$\|\delta z(k+1{:}p)\| \le \mu f_8(n) \|z(k+1{:}p)\| + \mathcal{O}(\mu^2)$$

where $f_i(n) = \mathcal{O}(n^2)$, $i = 5, 6$, and $f_i(n) = \mathcal{O}(n)$, $i = 7, 8$.

These results are as good as can be expected for any such procedure. As we state in the next section, we can expect sharp separation between singular subspaces associated with large and small singular values.

6.4.2 Perturbation Bounds for Invariant Subspaces

We consider in this section the effects of the bounds in Propositions 6.2 and 6.3 in the error in certain invariant subspaces of $\bar{B}$ resulting from Algorithms 6.1 and 6.2. Two perturbation results show that we expect that the subspaces for large and small singular values will be very accurately computed.

The componentwise backward error $\delta\bar{B}_0$ in Proposition 6.2 has a very small effect on the error in the subspaces. The following result is given for completeness.

PROPOSITION 6.4 ([39, LEMMA 4.5]). *Let*

$$B = \text{bidiag}(\gamma_1, \ldots, \gamma_n; \phi_1, \ldots, \phi_{n-1}) \tag{6.25}$$

and let

$$\tilde{B} = \text{bidiag}(\alpha_1 \gamma_1, \ldots, \alpha_n \gamma_n; \alpha_{n+1}\phi_1, \ldots, \alpha_{2n-1}\phi_{n-1}). \tag{6.26}$$

Let

$$\eta = \prod_{i=1}^{2n-1} \max\{\alpha_j, \alpha_j^{-1}\} - 1.$$

Let $w_1, \ldots, w_n$ *be the right singular vectors of* B *and let* $\tilde{w}_1, \ldots, \tilde{w}_n$ *be the right singular vectors of* $\tilde{B}$. *Let* $\sigma_1, \ldots, \sigma_n$ *be the singular values of* B *and define*

$$\rho_i = \min\left\{ 2, \min_{j \neq i} \frac{|\sigma_j - \sigma_i|}{|\sigma_i|} \right\}, \quad i = 1, 2, \ldots, n.$$

Let $Z_i = (w_1, \ldots, w_{i-1}, w_{i+1}, \ldots, w_n)$, *that is, the right singular vector matrix of* B *with its i-th column deleted. If* $\rho_i > \eta$, *then*

$$\|Z_i^T \tilde{w}_i\| \leq \sqrt{2}\left(\frac{\eta(1+\eta)}{\rho_i - \eta} + \frac{\eta}{2} \right). \tag{6.27}$$

Thus the effect of the relative errors $\delta\bar{B}_0$ on the updated matrix $\bar{B}$ is minimal and has little effect on the singular subspaces.

Proposition 6.3 implies that

$$\|\delta B_1\| \le \delta_B \|B_1\| + \mathcal{O}(\mu^2) \tag{6.28}$$

$$\|\delta B_2\| \le \delta_B \epsilon + \mathcal{O}(\mu^2) \tag{6.29}$$

$$\|\delta z(1:k)\| \le \delta_z \|z(1:k)\| + \mathcal{O}(\mu^2) \tag{6.30}$$

$$\|\delta z(k+1:p)\| \le \delta_z \|z(k+1:p)\| + \mathcal{O}(\mu^2). \tag{6.31}$$

Here, $\delta_B \le f_B(n)\mu$ and $\delta_z \le f_z(n)\mu$ where $f_B(n) = \mathcal{O}(n^2)$ and $f_z(n) = \mathcal{O}(n)$.

PROPOSITION 6.5. *Let $\bar{B}$ and $\bar{B} + \delta\bar{B}$ be diagonal matrices such that*

$$\bar{B} = \begin{pmatrix} \overset{k}{\bar{B}_1} & \overset{n-k}{\bar{\phi}_{k+1}e_k e_1^T} \\ 0 & \bar{B}_2 \end{pmatrix} \begin{matrix} k \\ n-k \end{matrix} , \quad \delta\bar{B} = \begin{pmatrix} \overset{k}{\delta\bar{B}_1} & \overset{n-k}{0} \\ 0 & \delta\bar{B}_2 \end{pmatrix} \begin{matrix} k \\ n-k \end{matrix} ,$$

where

$$\|\delta\bar{B}_1\| \le \delta_B \|\bar{B}_1\| + \mathcal{O}(\mu^2), \quad \|\delta\bar{B}_2\| \le \delta_B \epsilon + \mathcal{O}(\mu^2), \quad \left\| \begin{pmatrix} \bar{\gamma}_{k+1}e_1^T \\ \bar{B}_2 \end{pmatrix} \right\| \le \epsilon.$$

Let $\sigma_1 \ge \cdots \ge \sigma_k > \epsilon > \sigma_{k+1} \ge \cdots \ge \sigma_n$. Let $W, \widetilde{W} \in \mathcal{R}^{n \times n}$ be the matrices of right singular vectors of B and $B + \delta B$, respectively. If

$$W = (W_1 \ W_2), \quad \widetilde{W} = (\widetilde{W}_1 \ \widetilde{W}_2), \tag{6.32}$$

where W_1, $\widetilde{W}_1 \in \mathcal{R}^{n \times k}$, and W_2, $\widetilde{W}_2 \in \mathcal{R}^{n \times (n-k)}$, then

$$\|\widetilde{W}_1^T W_2\|_F \leq 2\delta_B \|B_1\| \|\bar{B}_1^{-1}\| \frac{\sigma_{k+1} + \sqrt{2}\,\epsilon}{\sigma_k - \sigma_{k+1}} + \mathcal{O}(\|\delta\bar{B}\|^2). \tag{6.33}$$

Proof. Let $\widetilde{w}_i$, $i = 1, 2, \ldots, k$ be the i-th right singular vector of $\bar{B} + \delta\bar{B}$ and let w_j, $j = k+1, \ldots, n$ be the j-th singular vector of $\bar{B}$. Then from standard perturbation bounds on the eigenvectors of $\bar{B}^T \bar{B}$ we have

$$|\widetilde{w}_i^T w_j| = \frac{|w_i^T \delta\bar{B}^T \bar{B} w_j + w_i^T \bar{B}^T \delta\bar{B} w_j|}{\sigma_i^2 - \sigma_j^2} + \mathcal{O}(\|\delta\bar{B}\|^2) \tag{6.34}$$

$$\leq \frac{\|\delta\bar{B}w_i\|\sigma_j + \|\delta\bar{B}w_j\|\sigma_i}{\sigma_i^2 - \sigma_j^2} + \mathcal{O}(\|\delta\bar{B}\|^2) \tag{6.35}$$

We now bound $\|\delta\bar{B}w_i\|$ for $i = 1, 2, \ldots, n$. First, let

$$w_i = \begin{pmatrix} w_i^{(1)} \\ w_i^{(2)} \end{pmatrix} \begin{matrix} k \\ n-k \end{matrix} \tag{6.36}$$

Then we have

$$\begin{pmatrix} \bar{B}_1 & \bar{\phi}_{k+1} \\ 0 & \bar{B}_2 \end{pmatrix} \begin{pmatrix} w_i^{(1)} \\ w_i^{(2)} \end{pmatrix} = \sigma_i \begin{pmatrix} y_i^{(1)} \\ y_i^{(2)} \end{pmatrix} = \sigma_i y_i$$

where y_i is the corresponding left singular vector. Thus we have

$$w_i^{(1)} = B_1^{-1}(\sigma_i y_i^{(1)} - \bar{\phi}_{k+1} w_i^{(2)}).$$

Therefore,

$$\|w_i^{(1)}\| = \|B_1^{-1}\|(\sigma_i + |\bar{\phi}_{k+1}|) \leq \|B_1^{-1}\|(\sigma_i + \epsilon). \tag{6.37}$$

Now we can say that

$$\|\delta\bar{B}w_i\|^2 \leq \|\delta\bar{B}_1 w_i^{(1)}\|^2 + \|\delta\bar{B}_2 w_i^{(2)}\|^2$$

which leads to

$$\|\delta\bar{B}w_i\| \leq \delta_B \|\bar{B}_1\| \|\bar{B}_1^{-1}\|(\sigma_i + \epsilon)^2 + \delta_B^2 \epsilon^2. \tag{6.38}$$

Equation (6.38) leads to

$$\|\delta\bar{B}w_i\| \leq \delta_B(\sigma_i + \sqrt{2}\,\epsilon)\|B_1\| \|\bar{B}_1^{-1}\|. \tag{6.39}$$

Combining (6.35) and (6.39) yields

$$|\tilde{w}_i^T w_j| \leq \delta_B \|B_1\| \|\bar{B}_1^{-1}\| \frac{(\sigma_i + \sqrt{2}\,\epsilon)\,\sigma_j + (\sigma_j + \sqrt{2}\,\epsilon)\,\sigma_i}{\sigma_i^2 - \sigma_j^2} + \mathcal{O}(\|\delta\bar{B}\|^2)$$

which is bounded by

$$|\tilde{w}_i^T w_j| \leq \delta_B \|B_1\| \|\bar{B}_1^{-1}\| \frac{(\sigma_i + \sqrt{2}\,\epsilon)}{\sigma_i - \sigma_j} + \mathcal{O}(\|\delta\bar{B}\|^2). \tag{6.40}$$

Thus for all $i = 1, 2, \ldots, k$ and $j = k + 1, \ldots, n$, we have

$$|\tilde{w}_i^T w_j| \leq \delta_B \|B_1\| \|\bar{B}_1^{-1}\| \frac{(\sigma_{k+1} + \sqrt{2}\,\epsilon)}{\sigma_k - \sigma_{k+1}} + \mathcal{O}(\|\delta\bar{B}\|^2). \tag{6.41}$$

which is now independent of i and j. The use of the Frobenius norm on $\widetilde{W}_1 = (\widetilde{w}_1, \ldots, \widetilde{w}_k)$ and $W_2 = (w_{k+1}, \ldots, w_n)$ yields (6.33). $\square$

Proposition 6.5 implies that the updating algorithm will always yield accurate subspaces for the first k and last $n - k$ singular values. For Algorithm 6.2 we must also bound the effect of δz which is qualitatively slightly different.

PROPOSITION 6.6. *Let $\bar{B}$ and W be an in Proposition 6.5 and let it satisfy $\bar{B}^T \bar{B} = \bar{V} B^T B \bar{V}^T - z z^T$ where $V \in \mathcal{R}^{n \times n}$ is the orthogonal matrix from Algorithm 6.2. Let $\widetilde{B}$ satisfy $\widetilde{B}^T \widetilde{B} = \bar{V} B^T B \bar{V}^T - (z + \delta z)(z + \delta z)^T$ and let δz have the form (6.30)-(6.31). Let $z = (x^T \ y^T)^T$, $x \in \mathcal{R}^k$, $y \in \mathcal{R}^{n-k}$ and assume that $\|y\| \le \epsilon$. Let $\widetilde{W} \in \mathcal{R}^{n \times n}$ be the matrix of right singular vectors of $\widetilde{B}$ and define $\widetilde{W}_1$ and $\widetilde{W}_2$ from (6.32). Then*

$$\|\widetilde{W}_1^T W_2\|_F \ \le \ \delta_z \sqrt{k(n-k)} \, \omega \, \bar{\kappa} \, \|B_1\| \, \|\bar{B}_1^{-1}\| \times$$

$$\left[\frac{\sigma_k + 3\epsilon}{\sigma_k - \sigma_{k+1}} + \frac{4\epsilon^2}{\sigma_k^2 - \sigma_{k+1}^2} \right] + \mathcal{O}(\|\delta z\|^2) \qquad (6.42)$$

where

$$\omega = \|\bar{B}^{-T} x\|, \quad \bar{\kappa} = \|B_1\| \, \|\bar{B}_1^{-1}\| \qquad (6.43)$$

Proof. Let $\widetilde{w}_i$, $i = 1, 2, \ldots, k$ be the i-th right singular vector of $\bar{B} + \delta \bar{B}$ and let w_j, $j = k+1, \ldots, n$ be the j-th singular vector of $\bar{B}$. Then from standard perturbation bounds on the eigenvectors of $\bar{B}^T \bar{B}$ we have

$$|\widetilde{w}_i^T w_j| \ = \ \frac{|w_i^T \delta z z^T w_j + w_i^T z \delta z^T w_j|}{\sigma_i^2 - \sigma_j^2} + \mathcal{O}(\|\delta z\|^2) \qquad (6.44)$$

$$\leq \frac{|\delta z^T w_i| \, |z^T w_j| + |\delta z^T w_j| \, |z^T w_i|}{\sigma_i^2 - \sigma_j^2} + \mathcal{O}(\|\delta z\|^2). \qquad (6.45)$$

If we use the partitioning of w_i in (6.36), then we have

$$|z^T w_i| = |x^T w_i^{(1)}| + |y^T w_i^{(2)}| = |x^T \bar{B}^{-1} \bar{B}_1 w_i^{(1)}| + |y^T w_i^{(2)}|$$

which means that

$$|z^T w_i| \leq \|\bar{B}^{-T} x\| \, \|\bar{B}_1 w_i^{(1)}\| + \epsilon \leq \omega \, \sigma_i + \epsilon, \quad i = 1, \ldots, n. \qquad (6.46)$$

We also have that

$$|\delta z^T w_i| \leq |\delta x^T w_i^{(1)}| + |\delta y^T w_i^{(2)}| \leq \delta_z \|x\| \, \|w_i^{(1)}\| + \|y\| \, \|w_i^{(2)}\|.$$

Using the facts that $\|x\| \leq \|B_1\|$ and $\|y\| \leq \epsilon$ and from (6.37), we have

$$|\delta z^T w_i| \leq \delta_z (\|B_1\| \, \|\bar{B}_1^{-1}\| \sigma_i + \epsilon) + \epsilon$$

which we simplify to

$$|\delta z^T w_i| \leq \delta_z \|B_1\| \, \|\bar{B}_1^{-1}\| (\sigma_i + 2\epsilon). \qquad (6.47)$$

Combining (6.45) with (6.46) and (6.47) yields

$$|\widetilde{w}_i^T w_j| \leq \delta_z \omega \, \|B_1\| \, \|\bar{B}_1^{-1}\| \frac{(\sigma_i + 2\epsilon)(\sigma_j + \epsilon) + (\sigma_j + 2\epsilon)(\sigma_i + \epsilon)}{\sigma_i^2 - \sigma_j^2}$$

$$+ \mathcal{O}(\|\delta z\|^2) \qquad (6.48)$$

$$\leq \quad \delta_z \omega \, \|B_1\| \, \|\bar{B}_1^{-1}\| \left[\frac{\sigma_j + 3\epsilon}{\sigma_i - \sigma_j} + \frac{4\epsilon^2}{\sigma_i^2 - \sigma_j^2} \right] + \mathcal{O}(\|\delta z\|^2). \qquad (6.49)$$

We note that this for $i = 1, 2, \ldots, k$ and $j = k + 1, \ldots, n$, thus we can bound (6.49) by

$$|\tilde{w}_i^T w_j| \leq \delta_z \omega \, \|B_1\| \, \|\bar{B}_1^{-1}\| \left[\frac{\sigma_k + 3\epsilon}{\sigma_k - \sigma_{k+1}} + \frac{4\epsilon^2}{\sigma_k^2 - \sigma_{k+1}^2} \right] + \mathcal{O}(\|\delta z\|^2). \qquad (6.50)$$

The bound (6.42) is obtained by computing the Frobenius norm of $\widetilde{W}_1^T W_2$, where $\widetilde{W}_1 = (\tilde{w}_1, \ldots, \tilde{w}_k)$ and $W_2 = (w_{k+1}, \ldots, w_n)$. $\quad\square$

6.5   Numerical Examples

In this section, we present a few examples from numerical experiments. These tests were performed using MATLAB on a SPARCstation 5 in IEEE Standard double precision with machine precision $\approx 10^{-16}$. As in Chapter 4 the algorithm employs the sliding window technique from signal processing.

At each step of the sliding window method with the window size m_0, an $m_0 \times n$ data matrix is constructed from an $m \times n$ observation matrix A by adding a new row to the data matrix in the previous window and deleting the oldest row from it. In step j, the row $m_0 + j$ of the observation matrix

Then Algorithms 6.1 and 6.2 take the diagonal matrix and the orthogonal matrix (right part) as initial input and the modifying vector r, and successively modifies these matrices at every window step. The vector $z = V^T r$ is computed at the beginning of each window step.

The one-sided Jacobi method [36] was used to compute the SVD of the initial window matrix $A^{(0)}$, which consists of the first m_0 rows of A, and to compare with our algorithms for rank estimation and the accuracy of the subspaces.

We tested our algorithms in the context of the total least squares (TLS) problems. See Section 2.7 for details. We used the TLS solutions via the Jacobi SVD as reference in checking the accuracy of the solution and rank estimations of our algorithms.

In Fig. 6.5-6.7, the rank estimated by our algorithms (solid line) and the true rank (dotted line but not visible in the plot) are given in the first plot. The horizontal axis represents the window steps and the vertical axis the numerical rank of the window matrix.

Let $W^{(j)}$ and $V^{(j)}$ be the right singular vector matrices computed by the Jacobi method and Algorithms 6.1 and 6.2, respectively. Then, using the Definition 2.3,

$$\theta_j = \|W_1^{(j)^T} V_2^{(j)}\|, \quad j = 1, 2, \ldots$$

where $W^{(j)} = (W_1^{(j)} \ W_2^{(j)})$ and $V^{(j)} = (V_1^{(j)} \ V_2^{(j)})$. We plot $\log_{10}(\sin(\theta_j))$ in the second plot of each figure.

Finally, the TLS errors

$$\tau_j = \frac{\|x_j - \tilde{x}_j\|}{\|x_j\|}$$

are given in logarithm in the last plot. Here, x_j and $\tilde{x}_j$ are the TLS solutions using the Jacobi method and our algorithms, respectively.

EXAMPLE 6.4. A, a 100-by-5 random matrix, b, a 100-by-1 random vector. Entries of A and b were chosen from a uniform distribution on the interval $(0, 1)$. 75 randomly chosen rows of $[A; b]$ were multiplied by $\gamma = 10^{-4}$ in order to vary the rank of the matrix, and $tol = 10^{-2}$. The window size p used was 10.

The first plot shows that our algorithms estimated the numerical ranks correctly throughout the sliding window steps in spite of frequent rank changes. The errors in tiny singular values were relatively large, and our algorithms almost always overestimated small singular values. However, they were close enough for the correct rank estimation.

The second plot in each figure shows that the noise subspace error is very small giving accurate TLS solutions.

EXAMPLE 6.5. Same as Example 6.4 except that $\gamma = 10^{-9}$ and $tol = 10^{-7}$.

EXAMPLE 6.6. Same as Example 6.4 except that the matrix had an outlier of size 10^5 at $(15, 1)$ position.

Both TLS solution errors and the noise subspace errors show that our algorithms give very accurate approximation to the subspaces under consideration. Moreover, the algorithm performs well even when some of tiny singular values become almost zero (indicated by '*' in the first plot). We tested several other examples, and these results were typical.

Since our downdating procedures use LINPACK downdating algorithm, it is not difficult to generate the cases where the algorithm breaks down when $\|a\| > 1$, for instance, when deleting a large row relative to γ (see Fig. 6.6) or a row that contains outliers (see Fig. 6.7). We used the CSNE approach in (6.14)–(6.17), and indicated these steps by '+' in the first plot.

The CSNE approach was used in all three examples and most extensively in Example 6.6 when downdating a row with an outlier. However, the performance of our algorithm was less satisfactory for the larger outlier.
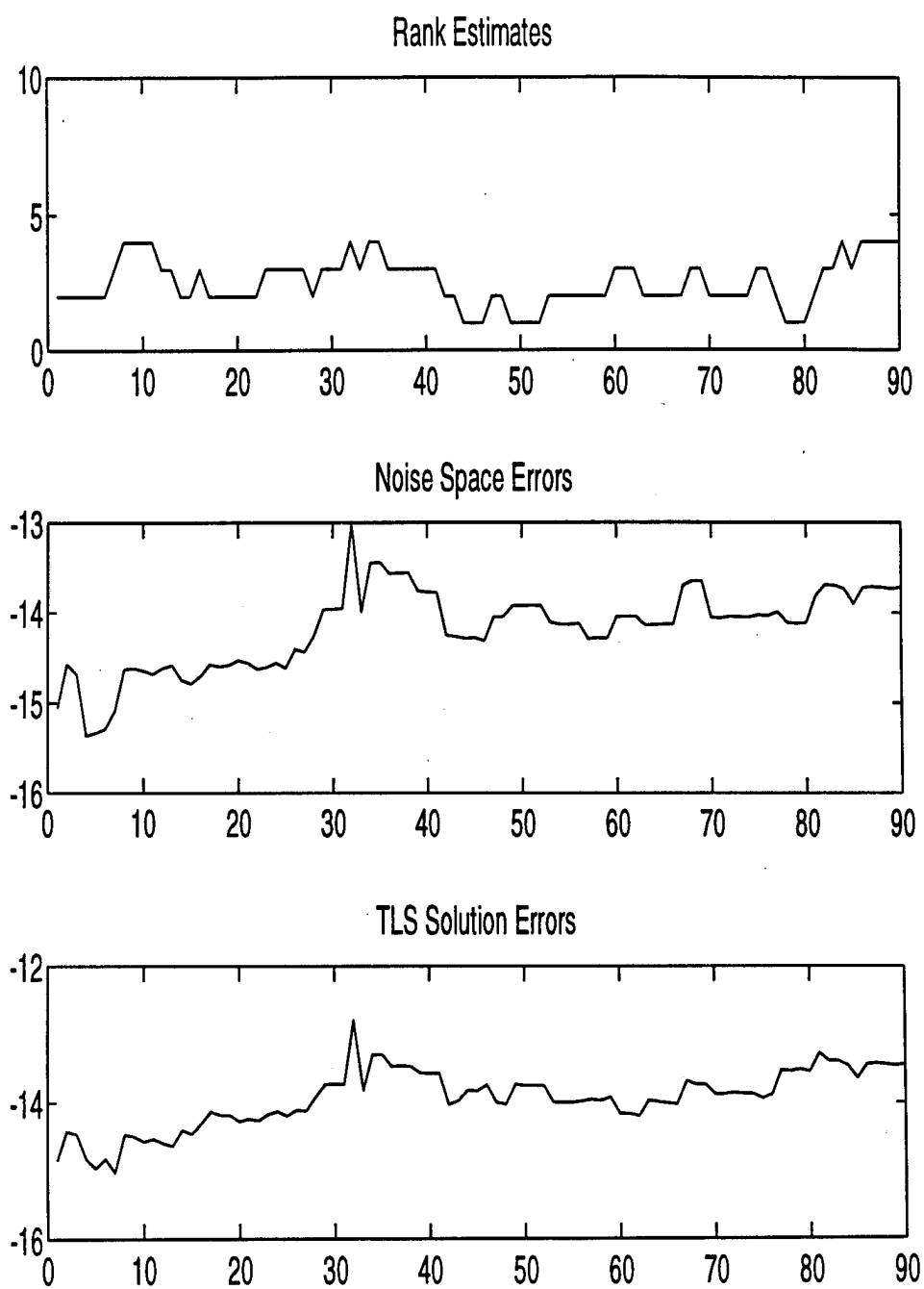
Rank Estimates

Noise Space Errors

TLS Solution Errors

Fig. 6.5. Example 6.4

Rank Estimates

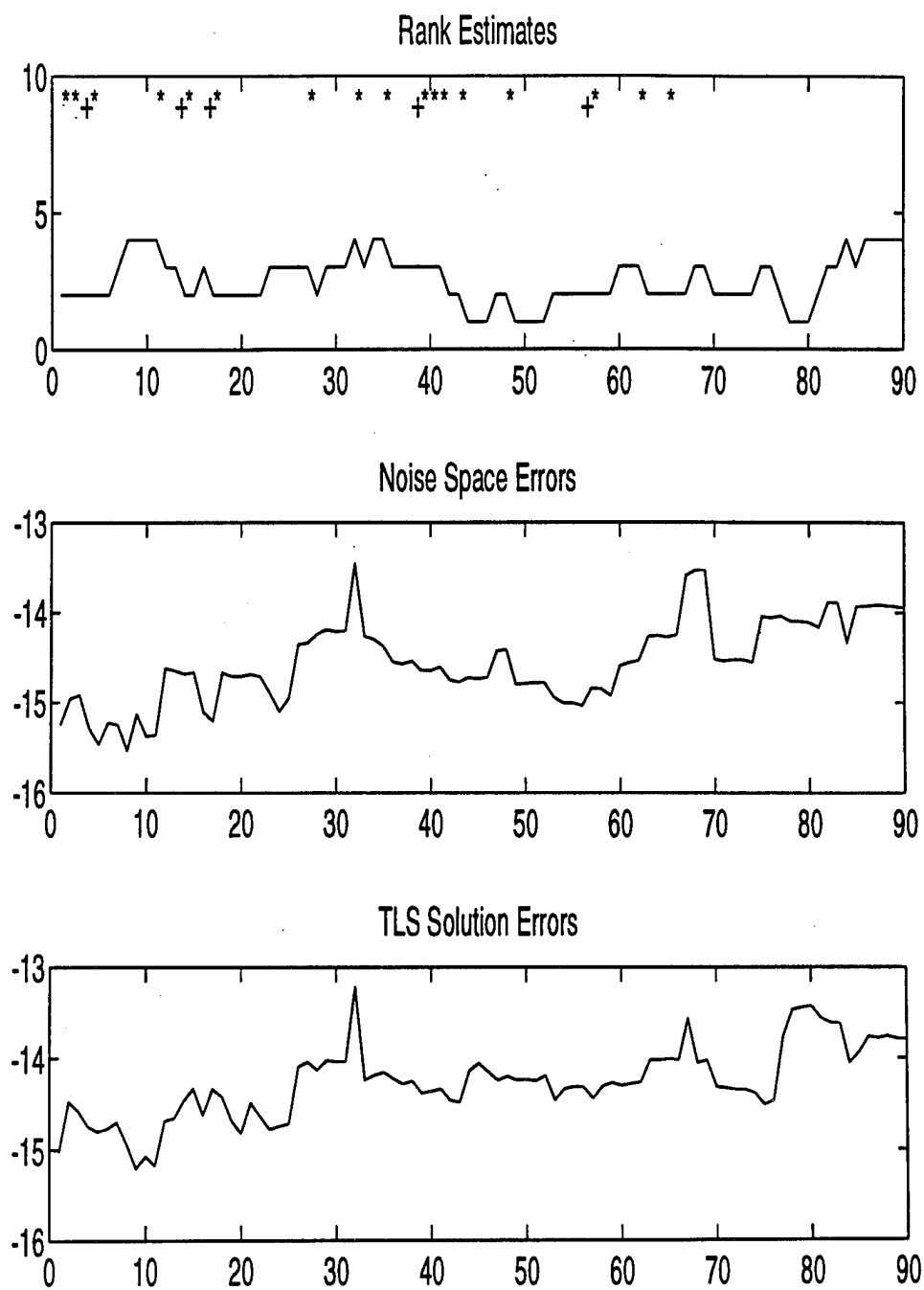## Noise Space Errors

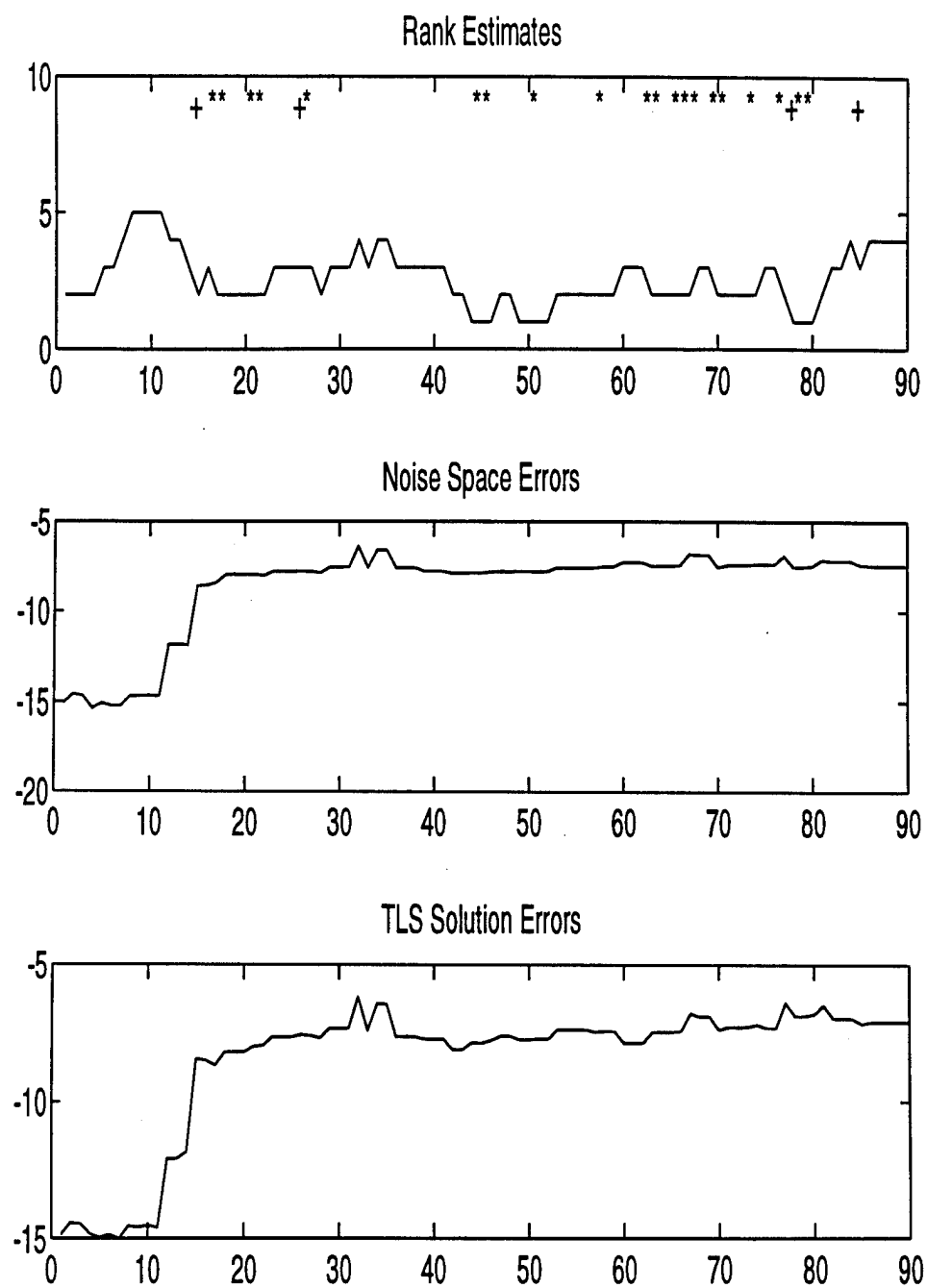## TLS Solution Errors

Fig. 6.6. Example 6.5

Fig. 6.7. Example 6.6

Chapter 7

Parallel Implementation

7.1 Introduction

In this chapter we describe a fully parallel bidiagonal reduction procedure for modifying the SVD. In Chapter 6 we showed that blockwise algorithms produced more accurate subspaces than the ordinary one-way chasing algorithms which ignore the block structure of the diagonal matrix. A VLSI implementation of the similar chasing schemes for the bidiagonal reduction for updating was also described in [1, 117], but without considering the large and small structure of the matrix. In this section we implement our algorithm on a distributed-memory MIMD multiprocessor. Two storage schemes are considered: cyclic storage scheme and consecutive storage scheme. We will show that the consecutive storage scheme implements the bidiagonal reduction much more efficiently.

The main idea behind the Algorithms 6.1 and 6.2 is to reduce the entries of the vectors x and y in opposite order and to chase the bulge in opposite direction, upper-left corner for the large block and lower-right corner for the small block. This is based on the two-way chasing scheme [125], which was also used in [116] with $k = n/2$ in the context of updating. The algorithm simply reduces the large and small blocks to almost bidiagonal form (see the 12-th matrix in Fig. 6.2) by ordinary chasing scheme, and uses 2×2 updating and downdating algorithms to eliminate x_k and y_1, followed by one step of the **qd** procedure on the small block to reduce it to the upper bidiagonal matrix. The

entire reduction steps for Algorithm 6.1 requires

$$\overbrace{k(k-1)+(n-k)(n-k-1)}^{\text{Step 1}}+\overbrace{(n-k-1)}^{\text{Step 3}}$$

$$= n^2 - 2nk + 2k^2 - (k+1) \tag{7.1}$$

plane rotations.

The algorithm allows simultaneous bidiagonal reductions on both large and small blocks, B_1 and B_2 (Step 1 in Algorithm 6.1) since they do not share any data throughout the reduction steps. Following similar notations used in [1], we denote $F_{i,j}$ as Givens plane rotations operating on rows i and j (left rotations), and $G_{i,j}$ as those operating on columns i and j (right rotations). Then from the dependency graph of this algorithm depicted in Fig. 7.1, we see that $G_{1,2}$ and $G_{n-1,n}$, the first rotations for each block, can be executed in parallel, and the sequence of the rotations that follows are also carried out in parallel. Hence, the whole reduction only takes

$$2 + \overbrace{3 + \cdots + 3}^{k-3} + 2(k-1) = 5k - 9 \quad \text{if } k \geq \lfloor \tfrac{5n+1}{10} \rfloor \tag{7.2}$$

$$2 + \overbrace{3 + \cdots + 3}^{k-3} + 2(n-k-1) + 1 = 5n - 5k - 8 \quad \text{if } k < \lfloor \tfrac{5n+1}{10} \rfloor \tag{7.3}$$

time steps. Obviously, the algorithm achieves an optimal performance when $k \approx n/2$.

7.2   Overview of Connection Machine

A Connection Machine (CM-5) system can have up to 16K physical processors or processing nodes (PNs). The CM-5 has two interprocessor communications networks: data network and control network. The control network is used for global operations
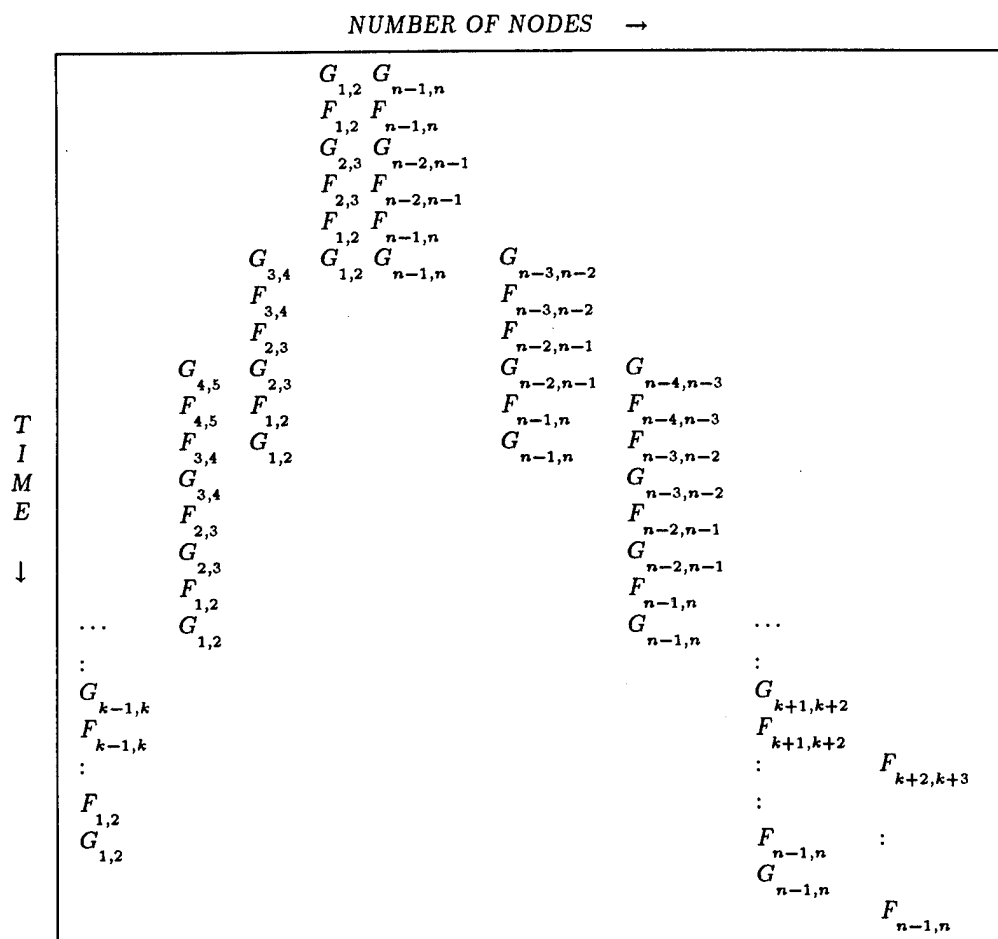
NUMBER OF NODES →

$$
\begin{array}{ccccccc}
 & & G_{1,2} & G_{n-1,n} & & & \\
 & & F_{1,2} & F_{n-1,n} & & & \\
 & & G_{2,3} & G_{n-2,n-1} & & & \\
 & & F_{2,3} & F_{n-2,n-1} & & & \\
 & & F_{1,2} & F_{n-1,n} & & & \\
 & G_{3,4} & G_{1,2} & G_{n-1,n} & G_{n-3,n-2} & & \\
 & F_{3,4} & & & F_{n-3,n-2} & & \\
 & F_{2,3} & & & F_{n-2,n-1} & & \\
 G_{4,5} & G_{2,3} & & & G_{n-2,n-1} & G_{n-4,n-3} & \\
 F_{4,5} & F_{1,2} & & & F_{n-1,n} & F_{n-4,n-3} & \\
 F_{3,4} & G_{1,2} & & & G_{n-1,n} & F_{n-3,n-2} & \\
 G_{3,4} & & & & & G_{n-3,n-2} & \\
 F_{2,3} & & & & & F_{n-2,n-1} & \\
 G_{2,3} & & & & & G_{n-2,n-1} & \\
 F_{1,2} & & & & & F_{n-1,n} & \\
 G_{1,2} & & & & & G_{n-1,n} & \\
\end{array}
$$

T
I
M
E

↓

T
I
M
E

↓

... $G_{1,2}$ $G_{n-1,n}$ ...

$\vdots$

$G_{k-1,k}$ $G_{k+1,k+2}$

$F_{k-1,k}$ $F_{k+1,k+2}$

$\vdots$ $\vdots$ $F_{k+2,k+3}$

$F_{1,2}$ $\vdots$

$G_{1,2}$ $F_{n-1,n}$ $\vdots$

$G_{n-1,n}$

$F_{n-1,n}$

Fig. 7.1.   Dependency Graph of the Modified Bidiagonal Reduction

such as synchronization and broadcasting. The data network, which uses, so-called, *4-ary fat tree* [72], supports operations for data transfers from a single source to a single destination.

The CM-5 supports both SIMD (Single Instruction Multiple Data) and MIMD (Multiple Instruction Multiple Data) programming models [66]. In the SIMD model, the data parallel programming associates one PN with each element of a data set. All PNs execute identical operations, each operating on data stored in its local memory, accessing data stored in the local memory of other PNs, or receiving data from the host computer.

In the MIMD programming model, each node has its own copy of the same program called node program, and executes the program asynchronously. The communication between the nodes is usually done by utilizing a set of efficient communication routines contained in the CM message-passing library, CMMD. For our implementation we chose the MIMD model because the rotations at each time step are different in terms of their types (left or right) and the data required.

In CM-5 a packet of size 20 bytes is used for nodal communications. First four bytes are used for control purposes and the rest of 16 bytes contain the data. If a packet is full, that is, if it contains 16 bytes of user data, the overhead of processing it is smaller than the message of different sizes. Therefore, the communication overhead will be smaller if a user made the message size a multiple of 16 bytes [90].

Moreover, a cluster is composed of four processing nodes, and the nodes with the same cluster share a common switching node capable of four times the bandwidth of the node at the leaf level. A Similar statement is true for the nodes as progressing toward the root. Each node must go through at least one switching node to communicate with the other node. To communicate with the node in a different cluster, the communication path will be longer. Hence, it takes longer to transfer the data to the node within the same cluster than to the node outside the cluster.

We assume the CM-5 consists of p processing nodes, and denote them as NODE(0), ..., NODE($p-1$). Here, we assume that p is a power of two. We choose one of the CMMD programming styles, Host/Node model, where the host processor allocates the data to the nodes, and collects the results for the analysis. The host program calls the node program residing in each node for the various tasks, and each node has identical node program. Once the node program is loaded in each node, it can be executed asynchronously. Both host and node programs for our implementation are written in FORTRAN.

7.3 Implementation Details

In this section we give a detailed description of parallel implementation of the Algorithm 6.1. First, we need the following definition.

DEFINITION 7.1. *A pair of left rotations* $F_{i,i+1}$ *and* $F_{j,j+1}$ *or right rotations* $G_{i,i+1}$ *and* $G_{j,j+1}$ *is said to be adjacent if* $|i - j| = 2$.

We also use LRED(i) to denote the sequence of plane rotations for eliminating x_i and restoring the resulting matrix into the upper bidiagonal matrix (large block), and SRED(i) to denote those for eliminating $y_{n-k-i+1}$ and restoring the matrix into the lower bidiagonal form (small block), that is,

$$\text{LRED}(i) = \{G_{i,i+1}, F_{i,i+1}, \ldots, F_{1,2}, G_{1,2}\}, \quad i = 1, \ldots, k-1 \tag{7.4}$$

$$\text{SRED}(i) = \{G_{n-i,n-i+1}, F_{n-i,n-i+1}, \ldots, F_{n-1,n}, G_{n-1,n}\}, \tag{7.5}$$

$$i = 1, \ldots, n-k-1.$$

From the dependency graph shown in Fig. 7.1, we note that LRED(i) and SRED(i) will always start and complete at the same time step although SRED($n - k - 1$), the last sequence for the small block, will finish before LRED($k - 1$) when $k > n/2$. In this case

the **qd** step of $F_{k+2,k+3}, F_{k+3,k+4}, \ldots, F_{n-1,n}$ can proceed while the large block is being reduced.

There are at least two storage schemes that can be used for implementing the bidiagonal reduction: cyclic storage scheme and consecutive storage scheme.

7.3.1 Basic Procedures

In Table 7.1 we describe communication primitives for the node-to-node communications.

Table 7.1. Communication Primitives

Primitives	Description	CMMD Routines
send(*nodelist;outlist*)	Send variables in *outlist* to each node in *nodelist*	`CMMD_send_block`
recv(*i;inlist*)	Receive variables in *inlist* from $\text{NODE}(i)$	`CMMD_receive_block`
swap(*i;inlist;outlist*)	Exchange variables in *outlist* with *inlist* of $\text{NODE}(i)$	`CMMD_swap`
send_and_recv (*i,j;inlist;outlist*)	Send variables in *outlist* to $\text{NODE}(j)$ and receive variables in *inlist* from $\text{NODE}(i)$ simultaneously	`CMMD_send_and_receive`

Note that all of CMMD communication routines used are blocking version, that is, each node waits until it finish sending or receiving the data without proceeding to the next executable code. This ensures that each node carries out the rotation with correct data as we will see in the next section.

7.3.2 Cyclic Storage Scheme

Suppose we store the data rowwise, so that $\text{NODE}(i)$ contains all of nonzero entries on the row i such as the diagonal entry, and the nonzero entries created by the chasing steps. Then we immediately observe that this scheme would not give a full parallelism among the nodes. For instance, at time Step 6 in Fig. 7.1 (sixth matrix in Fig. 6.2), $G_{1,2}$ and $G_{3,4}$ cannot be executed in fully parallel fashion because $\text{NODE}(2)$ has all three elements in the second row, but $(2,3)$ entry is also used in processing $G_{3,4}$, so that $\text{NODE}(2)$ has to communicate with both $\text{NODE}(1)$ and $\text{NODE}(3)$. In fact, any adjacent pair of right rotations would cause similar difficulties when storing the data rowwise.

However, this problem can be completely avoided by storing the data columnwise. We show this using the following proposition.

PROPOSITION 7.1. *Suppose n is even. Then if $k \neq n/2$, the dependency graph for Algorithm 6.1 shown in Fig. 7.1 can contain no adjacent pair of left rotations at any time step.*

Proof. We only consider the reduction steps for large block since the same argument applies to the small block. Let $G_{i,i+1}$ and $G_{i+1,i+2}$ be the first rotations in $\text{LRED}(i)$ and $\text{LRED}(i+1)$, respectively. Then, we see that $G_{i+1,i+2}$ of $\text{LRED}(i+1)$ is executed after completing the rotations $G_{i,i+1}, F_{i,i+1}, F_{i-1,i}$ of $\text{LRED}(i)$, i.e., when $G_{i-1,i}$ of $\text{LRED}(i)$ is executed. So, their indices differ by two. Since the reduction patterns proceeds as $GFFGFG \cdots FG$, and the indices for the rotations decrease by one for every pair of F and G, the indices of subsequent rotations $\text{LRED}(i)$ and the ones in $\text{LRED}(i+1)$ differ by at least two. Therefore, it is impossible to have $F_{l,l+1}$ in $\text{LRED}(i)$ and $F_{j,j+1}$ in $\text{LRED}(i+1)$, where $|l-j| = 2$ at the same step. For $\text{LRED}(i)$ and $\text{LRED}(j)$, where $|i-j| > 2$, the result is more obvious. $\square$

If n is odd, we have the same result regardless of the value of k. In fact, if n is even and $k = n/2$, it is possible to have two adjacent left rotations, namely, $F_{k-1,k}$ and $F_{k+1,k+2}$ in the same time step. But, the reduction step in this case will have the following form:

$$\begin{pmatrix} \ddots & & & & & \\ & r & r & & & \\ \rightarrow & r & r & & & \\ \rightarrow & \widehat{r} & r & & & \\ \rightarrow & & & e & \widehat{e} & \\ \rightarrow & & & e & e & \\ & & & e & e & \\ & & & & & \ddots \end{pmatrix}$$

However, since there is no data dependency among the nodes, they can be executed in parallel. Therefore, $\text{NODE}(j)$ stores all of nonzero entries of j-th column including the entries of x and y, that is, we partition the set of nodes $\{0, 1, \ldots, p-1\}$ into

$$P_l = \{0, 1, \ldots, p_0 - 1\}, \quad P_s = \{p_0, p_0 + 1, \ldots, p - 1\} \tag{7.6}$$

where $p_0 q < k \leq (p_0 + 1)q$. Here, $q = n/p$.

$\text{NODE}(j)$, $j \in P_l$, stores columns $l \leq k$ where

$$j + 1 \equiv l \pmod{p_0}$$

$\text{NODE}(j)$, $j \in P_s$, stores columns $k < l \leq n$ where

$$j - p_0 + 1 \equiv (l - k) \pmod{p_0}$$

EXAMPLE 7.1. Suppose $p = 8$, $n = 16$, $k = 11$. Then $q = 2$, $P_l = \{0,1,2,3,4\}$ and $P_s = \{5,6,7\}$.

j	0	1	2	3	4	5	6	7
columns	1,6,11	2,7	3,8	4,9	5,10	12,15	13,16	14

7.3.2.1 Chasing Patterns

Before we consider designing the code for the reduction steps, we need to categorize possible chasing patterns. All of the reduction steps for the large block in Fig. 6.2 fall in one of the chasing patterns described in Fig. 7.2.

$$l_1 : \begin{pmatrix} a & 0 \\ b & 0 \\ 0 & b \\ \hat{x} & x \end{pmatrix} \Longrightarrow \begin{pmatrix} a & d \\ b & a \\ d & b \\ 0 & x \end{pmatrix} \qquad s_1 : \begin{pmatrix} b & 0 \\ 0 & b \\ 0 & a \\ y & \hat{y} \end{pmatrix} \Longrightarrow \begin{pmatrix} b & d \\ a & b \\ d & a \\ y & 0 \end{pmatrix}$$

$$l_2 : \begin{pmatrix} b & a \\ \hat{d} & b \end{pmatrix} \Longrightarrow \begin{pmatrix} b & a \\ 0 & b \end{pmatrix} \qquad s_2 : \begin{pmatrix} b & \hat{d} \\ a & b \end{pmatrix} \Longrightarrow \begin{pmatrix} b & 0 \\ a & b \end{pmatrix}$$

$$l_3 : \begin{pmatrix} b & a & \hat{d} \\ 0 & b & a \end{pmatrix} \Longrightarrow \begin{pmatrix} b & a & 0 \\ d & b & a \end{pmatrix} \qquad s_3 : \begin{pmatrix} a & b & 0 \\ \hat{d} & a & b \end{pmatrix} \Longrightarrow \begin{pmatrix} a & b & d \\ 0 & a & b \end{pmatrix}$$

$$l_4 : \begin{pmatrix} a & 0 \\ b & a \\ \hat{d} & b \end{pmatrix} \Longrightarrow \begin{pmatrix} a & d \\ b & a \\ 0 & b \end{pmatrix} \qquad s_4 : \begin{pmatrix} b & \hat{d} \\ a & b \\ 0 & a \end{pmatrix} \Longrightarrow \begin{pmatrix} b & 0 \\ a & b \\ d & a \end{pmatrix}$$

Fig. 7.2. Chasing Patterns

The patterns l_1-l_4 correspond to the large block, and s_1-s_4 to the small block. Note that patterns l_i is symmetric to s_i simply because the types of resulting bidiagonal

matrices and the chasing direction for each block are exactly opposite. Here, b's represent the diagonal entries, a's super or subdiagonal, and d's bulges. Then, we see that $\text{LRED}(i)$ and $\text{SRED}(i)$ have chasing patterns, respectively,

$$\{l_1, l_2, \overbrace{l_3, l_4, l_3, l_4, \ldots, l_3, l_4}^{2(i-1)}\} \text{ and } \{s_1, s_2, \overbrace{s_3, s_4, s_3, s_4, \ldots, s_3, s_4}^{2(i-1)}\}.$$

Note that $G_{1,2}$ has l_1 for $\text{LRED}(1)$ with a undefined and l_4 for $\text{LRED}(i)$, $i > 1$. Similarly, $G_{n-1,n}$ has s_1 for $\text{SRED}(1)$ with a undefined and s_4 for $\text{SRED}(i)$, $i > 1$.

7.3.2.2 Host Program

The host program distributes the data among the nodes, coordinates the order of the reductions, and initiates the reduction process. The CMMD routine `CMMD_distrib_to_nodes` provides efficient ways of allocating σ_i and z_i into the local variables `b` and `z` of $\text{NODE}(i)$.

Except for the $\text{LRED}(1)$ and $\text{SRED}(1)$, the host program initiates the subsequent reductions at every three time steps, i.e., $\text{LRED}(i)$ $\text{SRED}(n - i + 2)$ begin at time step $3 * (i - 1)$. This can be done by sending a signal to $\text{NODE}(i)$ and $\text{NODE}(n - i)$ as soon as receiving the message from $\text{NODE}(i - 2)$ which just finished processing the rotation $F_{i-2,i-1}$.

Upon the completion of Step 1 of Algorithm 6.1 by the nodes, the host program calls the subroutines which will perform 2×2 updating or downdating and one step of the **qd** process to complete the bidiagonal reduction. This step of the **qd** process is completely serial unless $k > n/2$. If it is the case, $\text{NODE}(k + 1)$, ..., and $\text{NODE}(n)$ can carry out the **qd** step while the reduction on the large block is performed.

7.3.2.3 Node Program

The CM-5 allows a single node program for all the nodes. However, we need to design the code so that a single program can handle multiple chasing patterns at the same time. Moreover, each chasing pattern needs the data from two or three nodes, and these nodes must know in advance what types of operations to perform. For instance, all patterns except for l_3 and s_3 require two nodes to do the job, but l_3 and s_3 need three columns, and so require three nodes to get involved in the computation. To this end, each node keeps a local variable op which is continually modified at every time step. The value of op of $\text{NODE}(i)$ is determined by $\text{NODE}(i + 1)$ for reducing the large block and by $\text{NODE}(i - 1)$ for reducing the small block as we will see shortly.

The node programs for the corresponding patterns are given in Table 7.2. A unique value of op is assigned to each segment of the node program. The value of op determines which operation each node should perform at a given time step, and each node executes only the part of the code marked by its current value of op .

The host program 'wakes up' the $\text{NODE}(i)$ and $\text{NODE}(n - i)$ always with op = 1 and op = 11, respectively, to start $\text{LRED}(i)$ and $\text{SRED}(i)$ because they begin with eliminating x_i and $y_{n-k-i+1}$. For instance, let us consider $\text{LRED}(3) = \{G_{3,4}, F_{3,4}, F_{2,3}, G_{2,3}, F_{1,2}, G_{1,2}\}$, which has chasing patterns $\{l_1, l_2, l_3, l_4, l_3, l_4\}$. Then, $\text{NODE}(3)$ receives the value of op = 1, and it immediately signals $\text{NODE}(4)$ with op = 5 to carry out the pattern l_1. Upon the completion of l_1, $\text{NODE}(3)$ and $\text{NODE}(4)$ increment their values of op by one to continue on to the next pattern l_2. At this point it is not necessary for $\text{NODE}(3)$ to signal $\text{NODE}(4)$ to specify the types of operations. When l_2 is completed, $\text{NODE}(3)$ again increments its value op by one and signals $\text{NODE}(2)$ with the updated value of op to start the rotation $F_{2,3}$ with pattern l_3. Then, $\text{NODE}(2)$ signals $\text{NODE}(3)$ with op = 7 and $\text{NODE}(4)$ with op = 9, and all three nodes execute parts of the code

Table 7.2. Node Programs Using the Cyclic Storage Scheme

Pattern	$\text{NODE}(i)$	$\text{NODE}(i+1)$	$\text{NODE}(i+2)$
l_1	/* op = 1 */ $\mathbf{swap}(i+1; a, b, x; \tilde{b}, \tilde{x})$ $\mathbf{formrot}(\tilde{x}, x, c, s)$ $a \leftarrow c * a$ $b \leftarrow c * b$ $d \leftarrow -s * \tilde{b}$	/* op = 5 */ $\mathbf{swap}(i; b, x; \tilde{a}, \tilde{b}, \tilde{x})$ $\mathbf{formrot}(x, \tilde{x}, c, s)$ $d \leftarrow -s * \tilde{a}$ $a \leftarrow s * \tilde{b}$ $b \leftarrow c * b$	
l_2	/* op = 2 */ $\mathbf{formrot}(b, d, c, s)$ $\mathbf{send}(i+1; c, s)$	/* op = 6 */ $\mathbf{recv}(i; c, s)$ $\mathbf{applyrot}(a, b, c, s, 1)$	
l_3	/* op = 3 */ $\mathbf{recv}(i+2; c, s)$ $b \leftarrow c * b$ $d \leftarrow s * b$	/* op = 7 */ $\mathbf{recv}(i+2; c, s)$ $\mathbf{applyrot}(b, a, c, s, 1)$	/* op = 9 */ $\mathbf{formrot}(a, d, c, s)$ $\mathbf{send}(i, i+1; c, s)$
l_4	/* op = 4 */ $\mathbf{swap}(i+1; a, b, d; \tilde{a}, \tilde{b})$ $\mathbf{formrot}(\tilde{b}, d, c, s)$ $a \leftarrow c * a$ $\mathbf{applyrot}(\tilde{a}, b, c, s, 1)$	/* op = 8 */ $\mathbf{swap}(i; a, b; \tilde{a}, \tilde{b}, \tilde{d})$ $\mathbf{formrot}(b, \tilde{d}, c, s)$ $d \leftarrow s * \tilde{a}$ $\mathbf{applyrot}(a, \tilde{b}, c, s, 1)$	
s_1	/* op = 11 */ $\mathbf{swap}(i+1; b, y; \tilde{a}, \tilde{b}, \tilde{y})$ $\mathbf{formrot}(y, \tilde{y}, c, s)$ $b \leftarrow c * b$ $a \leftarrow s * \tilde{b}$ $d \leftarrow s * \tilde{a}$	/* op = 15 */ $\mathbf{swap}(i; a, b, y; \tilde{b}, \tilde{y})$ $\mathbf{formrot}(\tilde{y}, y, c, s)$ $d \leftarrow -s * \tilde{b}$ $b \leftarrow c * b$ $a \leftarrow c * a$	
s_2	/* op = 12 */ $\mathbf{recv}(i+1; c, s)$ $\mathbf{applyrot}(a, b, c, s, 1)$	/* op = 16 */ $\mathbf{formrot}(b, d, c, s)$ $\mathbf{send}(i; c, s)$	
s_3	/* op = 13 */ $\mathbf{formrot}(a, d, c, s)$ $\mathbf{send}(i+1, i+2; c, s)$	/* op = 17 */ $\mathbf{recv}(i; c, s)$ $\mathbf{applyrot}(b, a, c, s, 1)$	/* op = 19 */ $\mathbf{recv}(i; c, s)$ $d \leftarrow s * b$
s_4	/* op = 14 */ $\mathbf{swap}(i+1; a, b; \tilde{a}, \tilde{b}, \tilde{d})$ $\mathbf{formrot}(b, \tilde{d}, c, s)$ $\mathbf{applyrot}(a, \tilde{b}, c, s, 1)$ $d \leftarrow s * \tilde{a}$	/* op = 18 */ $\mathbf{swap}(i; a, b, d; \tilde{a}, \tilde{b})$ $\mathbf{formrot}(\tilde{b}, d, c, s)$ $\mathbf{applyrot}(\tilde{a}, b, c, s, 1)$ $a \leftarrow c * a$	

according to the value of **op** . When this step is finished, $\text{NODE}(2)$ signals the host node to start $\text{LRED}(4)$, and it moves on to the next pattern l_4 which can be done exactly the same way. Similarly, we process the repeated patterns l_3 and l_4 in the same fashion.

The node program for the cyclic storage scheme is described in the Appendix.

7.3.3   Consecutive Storage Scheme

In this scheme consecutive blocks of the bidiagonal matrix are stored in each node. We partition the set of nodes exactly the same way as in (7.6) Using MATLAB notation, $\text{NODE}(j)$ stores

$$
\begin{aligned}
&jq + 1\!:\!(j+1)q && \text{if } j < p_0 - 1 \\
&(p_0 - 1)q + 1\!:\!k && \text{if } j = p_0 - 1 \\
&n\!:\!-1\!:\!n - q + 1 && \text{if } p_0 \le j < p - 1 \\
&n - (p - p_0 - 1)q\!:\!-1\!:\!k + 1 && \text{if } j = p - 1
\end{aligned}
$$

columns of B.

Denote $n_c(j)$ as the number of columns which the $\text{NODE}(j)$ has in its memory. Then, we have

$$
\begin{aligned}
n_c(j) \quad &= q, \quad i = 0, \ldots, p_0 - 2, p_0, p_0 + 1, \ldots, p - 2 \\
n_c(p_0 - 1) \quad &= (1 - p_0)q + k \\
n_c(p - 1) \quad &= (n - k) - (p - p_0 - 1)q.
\end{aligned}
$$

EXAMPLE 7.2. Suppose $p = 8$, $n = 16$, $k = 11$. Then $q = 2$, $P_l = \{0, 1, 2, 3, 4\}$ and $P_s = \{5, 6, 7\}$. Each node has the following columns:

j	0	1	2	3	4	5	6	7
columns	1,2	3,4	5,6	7,8	9,10,11	16,15	14,13	12
$n_c(j)$	2	2	2	2	3	2	2	1

Note that the last $n-k$ columns are stored in the nodes in P_s in reverse order. This ordering makes it possible for the nodes in P_l and P_s to have identical node programs for the bidiagonal reductions on the large and small blocks. From Fig. 6.2, we see that the bidiagonal reductions on these two blocks are exactly opposite, so that the reduction on the small block can be done by reversing the order of the diagonal entries and y, reducing the block exactly the same way as the large block, and again reversing the diagonal and bidiagonal entries when completed. Therefore, we only describe the reduction steps (6.8), that is, only for the nodes in P_l.

As in cyclic storage scheme NODE(j) contains variables a[i], b[i], z[i], i=1,..., $n_c(j)$ to store, respectively, the subdiagonal or superdiagonal entries, diagonal entries, and z_i. As mentioned before, the orders of array elements are reversed.

Following the notation in (7.4), NODE(0) initiates LRED(i), $i,\ldots,q-1$, sequentially. Then it needs to communicate with NODE(1) to execute the chasing pattern l_1 of LRED(q), reducing the entries of z. The rest of LRED(q) and starting LRED($q+1$), which is the responsibility of NODE(1), are done simultaneously. We repeat this process until all of the nodes will have finished their portion of the bidiagonal reduction.

Let us call a node which initiates a LRED(i) for some i at a given time, the *master* node. A *slave* node is the node which once became the master node, but now has task of chasing bulges as far as it can. Hence, at any time, there can exist only one master node, say, NODE(j), for some $j \leq p_0 - 1$, and j slave nodes, NODE(0), $\ldots$, NODE($j-1$). The rest of the nodes stay idle. Note that NODE($p_0 - 1$) never becomes a slave node.

A pseudo-code for the node program is given in Fig. 7.3.

```
/* We assume buf contains b[q-1], b[q], a[q], z[q]. */

/* Node(0) is the master */
if myid == 0 then
   for i = 1 to numcol-1
      reduce(i);
   end;
   send(myid+1, buf); recv(myid+1, buf); bchase(q, numcol);
   for i=q to k-q        /* Now it's a slave */
      send(myid+1, buf); recv(myid+1, buf); bchase(q, numcol);
   end;

/* Node(idhi) is the master */
else if myid == idhi then
   recv(myid-1, buf); redxnode(a, b, d, z, buf); send(myid-1, buf);
   for i = 1 to numcol-1
      reduce(i); recv(myid-1, buf); chxnode(a, b, d, buf);
      send(myid-1, buf);
   end;

/* Node(1), ... , Node(idhi-1) */
else
   recv(myid-1, buf); redxnode(a, b, d, z, buf);
   send(myid-1, buf);
   for i = 1 to numcol-1 /* It's now the master */
      reduce(i); recv(myid-1, buf); chxnode(a, b, d, buf);
      send(myid-1, buf);
   end;

   /* reduce across the node */
   send(myid+1, buf); recv(myid+1, buf); bchase(q, numcol);
   for i = (myid+1)*numcol to k-1 /* It's now a slave */
      send_and_recv(myid-1, myid+1, bufin, bufout);
      chxnode(a, b, d, bufin);
      send_and_recv(myid+1, myid-1, bufout, bufin);
      bchase(q, numcol);
   end;
end;
```

Fig. 7.3. Node Program for the Consecutive Storage Scheme

Here, `idhi` $= p_0 - 1$ and `numcol` $= n_c(j)$. The function `reduce(i)` performs the following transformation:

$$
\begin{pmatrix}
b_1 & a_1 & & & & & & \\
 & b_2 & a_2 & & & & & \\
 & & \ddots & \ddots & & & & \\
 & & & b_{i-1} & a_{i-1} & & & \\
 & & & & b_i & 0 & & \\
 & & & & & b_{i+1} & & \\
 & & & & \widehat{z}_i & z_{i+1} & &
\end{pmatrix}
\Longrightarrow
\begin{pmatrix}
b_1 & a_1 & & & & & & \\
 & b_2 & a_2 & & & & & \\
 & & \ddots & \ddots & & & & \\
 & & & b_{i-1} & \widetilde{a}_{i-1} & d & & \\
 & & & & \widetilde{b}_i & \widetilde{a}_i & & \\
 & & & & e & \widetilde{b}_{i+1} & & \\
 & & & & & \widetilde{z}_{i+1} & &
\end{pmatrix}
$$

followed by `bchase(i)`

$$
\begin{aligned}
&\begin{pmatrix}
b_1 & a_1 & & & & & \\
 & b_2 & a_2 & & & & \\
 & & \ddots & \ddots & & & \\
 & & & b_{i-1} & \widetilde{a}_{i-1} & d & \\
 & & & & \widetilde{b}_i & \widetilde{a}_i & \\
\rightarrow & & & & \widehat{e} & \widetilde{b}_{i+1} & \\
\rightarrow & & & & & \widetilde{z}_{i+1} &
\end{pmatrix}
\Longrightarrow \cdots \Longrightarrow
\begin{pmatrix}
\bar{b}_1 & \bar{a}_1 & & & & & \\
 & \bar{b}_2 & \bar{a}_2 & & & & \\
 & & \ddots & \ddots & & & \\
 & & & \bar{b}_{i-1} & \bar{a}_{i-1} & & \\
 & & & & \bar{b}_i & \bar{a}_i & \\
 & & & & & \bar{b}_{i+1} & \\
 & & & & & \widetilde{z}_{i+1} &
\end{pmatrix},
\end{aligned}
$$

`redxnode` carries out the sequence of pattern l_1-l_2-l_3, and `chxnode` l_3-l_4-l_3.

7.3.4 Computation Cost

Let τ_M, τ_A, and τ_S be, respectively, the time required for one multiplication, one addition, and one square root. Then the time required for (6.8) is

$$T(k) := kT_g + (k-1)T_a + (4k-4)\tau_M + \sum_{i=1}^{k-2} T_{red}(k-i+1) \tag{7.7}$$

where T_g, T_a and $T_{red}(i)$ are, respectively, the time required for **formrot**, **applyrot**, and **reduce(i)**. Since

$$T_g = 10\tau_M + 2\tau_A \tag{7.8}$$

$$T_a = 4\tau_M + 2\tau_A \tag{7.9}$$

$$T_{red}(j) = (2j-3)T_g + (2j-3)T_a + (4j-10)\tau_M, \tag{7.10}$$

the time required for reducing the large block to bidiagonal form is

$$T(k) = (16k^2 + 34k - 196)\tau_M + (4k^2 + 8k - 38)\tau_A \tag{7.11}$$

Moreover, 2 × 2 updating or downdating requires

$$T_{22} = \begin{cases} 12\tau_M + \tau_A + \tau_S & \text{for updating} \\ 9\tau_M + 6\tau_A + 6\tau_S & \text{for downdating} \end{cases} \tag{7.12}$$

and one step of the **qd** process needs

$$T_{qd} = (10n - 10k - 8)\tau_M + 2(n - k - 1)\tau_A. \tag{7.13}$$

Therefore, total cost required for the Algorithm 6.1 by each node, is, on average,

$$T_B := \frac{1}{p}\{(T(k) + T(n-k) + T_{22} + T_{qd})\} \tag{7.14}$$

Suppose $k = n/2$. Then

$$T_B = \frac{1}{p}\{(2T(k) + T_{22} + T_{qd})\} \tag{7.15}$$

$$= \frac{1}{p}\{(32k^2 + 78k - 390)\tau_M + (8k^2 + 18k - 75)\tau_A + 3\tau_S\} \tag{7.16}$$

Here we took the average for T_{22}.

7.3.5 Communication Cost

For our analysis, we ignore any possible contention problem for simplicity. Let α_d and β_d be, respectively, the startup time and the data transfer rate, time per byte. We need at most four `real*4` in single precision, so that sending this message of size 16 takes $\alpha_d + 4 * 4\beta_d$ seconds. Then the data in the packets used in communication would be of size multiple of 16, which makes the communication the most efficient on CM-5 [90].

7.3.5.1 Cyclic Storage Scheme

From Table 7.2, we see that F_{ij} and G_{ij} require communication at every window step. Hence, from (7.2) and (7.3), the communication cost becomes

$$T_{CYC} := \begin{cases} (5k - 9)(\alpha_d + 16\beta_d) & \text{if } k > n/2 \\ (5n - 5k - 8)(\alpha_d + 16\beta_d) & \text{if } k \leq n/2 \end{cases} \tag{7.17}$$

seconds, which is impractical on a coarse-grain distributed-memory MIMD multiprocessor due to enormous communication overhead. Moreover, very few computations are carried out by each node (see Table 7.2). Therefore, a fine-grain multiprocessor with fast communication ability would be required to make this storage scheme practical for bidiagonal reduction.

7.3.5.2 Consecutive Storage Scheme

Since NODE(0) initiates and terminates the bidiagonal reduction for the large block, the total communication cost for this scheme is determined by the number of **send** and **recv** calls done by NODE(0). For reducing the first $q-1$ columns of B, which is done by NODE(0), no communication is required. As a slave node NODE(0) needs to communicate $k-q+1$ times with NODE(1) to chase the bulges.

Moreover, when $k \leq n/2$, NODE(p_0) makes extra effort to complete bidiagonal reduction by applying one step of the **qd** process which requires one more communication step. When $k \geq n/2$, nodes in P_s can carry out the **qd** step while the large block gets reduced by the nodes in P_l. Therefore, the total communication cost for this storage scheme becomes

$$T_{CON} := \begin{cases} (k-q+2)(\alpha_d + 16\beta_d) & \text{if } k > n/2 \\ \{(n-k)-q+1+(|P_s|-1)\}(\alpha_d + 16\beta_d) & \text{if } k \leq n/2 \end{cases} \tag{7.18}$$

seconds. Here, $|P_s|$ denotes the number of nodes in P_s. Note that T_{CON} becomes worst when $k \leq n/2$ because the reduction on the large block is already completed before the **qd** process begins, and hence there is no parallelism among the nodes for this phase of the algorithm, requiring $|P_s|-1$ additional communication steps.

7.4 Timing Results

We implemented Algorithm 6.1 with both storage schemes on CM-5 that consists of up to 32 nodes at the Northeast Parallel Architectures Center (NPAC) at Syracuse University. Each node of the CM-5 is a SPARC chip which runs at 32 MHz and delivers 22 Mips and 5 Mflops. There is a 64 Kb instruction and data cache and a 16 Mb memory in each node [111]. In each node, there are two vector units; each vector unit is capable of peak rate 64 Mflops.

We generated an n-by-n random diagonal matrix B of the form (1.2) and a random n-vector z with $k = n/2$, where $n = 64, 128, 256, 512, 1024$. We only implemented the bidiagonal reduction part for modifying the SVD described in Algorithm 6.1. Computing the SVD of a general full matrix on a distributed-memory multiprocessor is described in [71], and computing the SVD of the bidiagonal matrix in [33].

The execution time for bidiagonal reduction using the consecutive storage scheme with different matrix sizes and various set of processing nodes is given in Table 7.3. Here, we use the following definition for *speed-up* achieved by our algorithm

$$S = \frac{T(1)}{T(p)}$$

where $T(p)$ is time required to execute the program on p processors. Similarly, we define the *efficiency* of a parallel algorithm as

$$E = \frac{T(1)}{p\,T(p)}.$$

We observe a linear speed-up as n increases with p fixed. When n is small, we cannot expect any speed-up mainly because of high communication cost compared to the computation cost. For instance, when $n = 64$, more than 50% of the total cost accounts

for the communication regardless of the number of processing nodes (except for the case $p = 4$). In fact, when $p = 32$, most of time were spent in communication. In this case, since the value of q is only two, each node has so little to do, and therefore the ratio of the time spent in communication and computation is quite large.

From (7.18) since $q = n/p$, we see that as p increases, so does T_{CON}. However, in general, q is small compared to k. Hence, the difference in the communication cost between different p is not significant as seen in Table 7.3. In general, as n increases, so does q, and therefore, the ratio of communication cost and total cost also decreases. However, when p is small and n is large, significant part of time were spent in communication ($p = 4$, $n = 512, 1024$; $p = 8$, $n = 1024$), and little speed-up was gained.

Table 7.4 shows the difference in execution time for various k when $n = 1024$ and $p = 32$. Clearly, we achieve an optimal speed-up when k is close to $n/2$. Note also that the bidiagonal reduction with $k > n/2$ is slightly faster than that with $k < n/2$ mainly because when $k > n/2$, the **qd** step can be executed in parallel with the last chasing step for the large block.

We also implemented Algorithms 6.1 and 6.2 using the cyclic storage scheme. However, it was embarrassingly slow, and no speed-up was gained in any case. As we have shown in the previous section, this storage scheme becomes totally impractical when $n > p$ due to high communication overhead caused by severe contention problems.

Table 7.3. CPU Time (in sec) for the Bidiagonal Reduction

p	n	Comp	Comm	Total	% Comm	Speedup
4	64	0.0182	0.0045	0.0227	19.8	3.3
	128	0.0746	0.0086	0.0831	10.3	3.5
	256	0.3032	0.0171	0.3203	5.3	3.6
	512	1.5543	0.1314	1.6857	7.8	2.7
	1024	5.2692	0.3672	5.6364	6.5	3.3
8	64	0.0079	0.0078	0.0157	49.7	4.8
	128	0.0338	0.0137	0.0475	28.8	6.1
	256	0.1397	0.0266	0.1665	16.0	6.8
	512	0.5697	0.0529	0.6308	8.4	7.2
	1024	3.4508	0.3929	3.8326	10.3	4.9
16	64	0.0045	0.0136	0.0181	75.0	4.2
	128	0.0214	0.0235	0.0449	52.3	6.5
	256	0.0929	0.0417	0.1346	31.0	8.5
	512	0.3878	0.0768	0.4645	16.5	9.8
	1024	1.2302	0.1363	1.3664	9.9	13.8
32	64	0.0013	0.0197	0.0210	93.8	3.6
	128	0.0089	0.0294	0.0383	76.8	7.6
	256	0.0429	0.0504	0.0934	54.0	12.3
	512	0.1875	0.0903	0.2778	32.5	16.4
	1024	0.7824	0.1537	0.9361	16.2	20.1

Table 7.4. CPU Time (in sec) with Various k ($n = 1024$, $p = 32$)

k	Total	Speedup	k	Total	Speedup
64	1.9039	9.9	960	1.8818	9.9
128	1.7741	10.6	896	1.7542	10.7
256	1.5138	12.4	768	1.4999	12.5
512	0.9361	20.1			

Chapter 8

Conclusion

We have presented efficient algorithms for modifying the TSODs, and shown that the TSOD can be updated and downdated in $\mathcal{O}(n^2)$ flops in a manner that preserves its structure. The backward error analysis and perturbation theory show that the procedures satisfy a blockwise stability property. Thus if our interest is in separating the two subspaces associated with the large and small singular values, we will obtain answers that are as good as can be expected. The use of this perturbation theory shows that we achieve more accuracy in the singular values and more orthogonality in the singular vectors that result from our update procedures. Our numerical tests show some improvement in the accuracy of downdated singular values using our algorithm instead of general chasing.

Our approach to modifying the ULVD is particularly promising. It is simple to implement for both updating and downdating, and preserves the rank-revealing structure often without the deflation process to compute the numerical rank of the matrix. Moreover, one can efficiently track the size of each block of lower triangular part of factorization for an accurate monitoring of the condition of downdating problem. Furthermore, data independence among the blocks makes the algorithms parallelizable.

We also have given algorithms for rank-one updates and downdates of the SVD and partially reduced bidiagonal forms. It has also been shown that these algorithms satisfy a blockwise stability criterion that has not been shown for other algorithms. The algorithms proposed to allow the user to specify a tolerance between large and small

singular values, and the separation between the associated subspaces is preserved by the algorithm.

Finally, we presented an efficient method for modifying the SVD in parallel to establish its practical value in real time applications. The algorithm preserves the block structures, maintaining the efficiency of parallel procedures as well. The consecutive storage scheme outperforms the cyclic storage scheme in bidiagonal reduction due to high communication cost of the latter scheme. The experiments show the efficiency in using the processors was slightly over 60% for the consecutive storage scheme.

Although the entire thesis is devoted to the problem of modifying the TSOD, there are a number of unresolved issues and problems. We suggest a few in the following:

- The stability of the CSNE approach taken when downdating is not possible is determined only by experimental results. In particular, one needs to know how good x in (6.17) really is. A rigorous error analysis should be performed by extending such results as those due to Björck [18].

- Parallelizing the ULVD procedures is much more challenging than the SVD procedures. Although the reduction on the large and small blocks can be executed simultaneously, it is difficult to enhance parallelism within the block. As suggested in [103] for parallelizing the URVD, we may require a fine-grain MIMD architecture for efficient implementation of systolic arrays.

- A procedure described by Gu and Eisenstat [56] is a promising approach to modify the singular vector matrix after modifying the SVD. They use an adaptive version of one-dimensional fast multipole method [29]. However, an efficient parallel implementation for this acceleration method is not, as yet, available although a parallel procedure for non-adaptive version has been developed [45]. Together with our

parallel bidiagonal reduction procedure described in Chapter 7, and parallel al-
gorithms for computing the SVD of bidiagonal matrices [33, 65, 107], the SVD
algorithm described in Chapter 6 can be implemented fully in parallel.

- It would also be interesting to extend algorithms for modifying the ULVD to those
  for modifying the ULLVD for two matrices [75] as an approximation to the gen-
  eralized SVD [84, 120]. We also need to analyze the stability and complexity of
  the algorithms for modifying the ULLVD and derive the error bounds on the sub-
  spaces computed by the ULLVD compared with those by the generalized SVD. The
  Estimator-Correlator array processor [108, 109] that can implement the estimator
  kernel using the ULVD and the inverse noise kernel using the ULLVD would be a
  practical application.

References

[1] A. Abdallah and Y. Hu. Parallel VLSI computing array implementation for signal subspace updating algorithm. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-37:742–748, 1989.

[2] G. Adams, M.F. Griffin, and G.W. Stewart. Direction-of-arrival estimation using the rank-revealing URV decomposition. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, Washington, D.C., 1991. IEEE.

[3] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The design and analysis of computer algorithms*. Addison-Wesley, Reading, MA, 1974.

[4] S.T. Alexander, C.-T. Pan, and R.J. Plemmons. Analysis of a RLS hyperbolic rotations scheme for signal processing. *Linear Algebra and Its Applications*, 98:3–40, 1988.

[5] L.P. Ammann. Robust image processing for remote sensing data. In M. Moonen and B. De Moor, editors, *SVD and signal processing III: algorithms, architectures, and applications*, pages 333–340, New York, 1995. Elsevier.

[6] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. DuCroz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK User's Guide*. SIAM Publications, Philadelphia, 1992.

[7] H.C. Andrews and C.L. Patterson. Singular value decomposition and digital image processing. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-24:26–53, 1976.

[8] J.L. Barlow. Stability analysis of the G-algorithm and a note on its application to sparse least squares problems. *BIT*, 25:507–520, 1985.

[9] J.L. Barlow. Error analysis and implementation aspects of deferred correction for equality constrained least-squares problems. *SIAM Journal on Numerical Analysis*, 25:1340–1358, 1988.

[10] J.L. Barlow. Error analysis of update methods for the symmetric eigenvalue problem. *SIAM Journal on Matrix Analysis and Applications*, 14:598–618, 1993.

[11] J.L. Barlow and J.W. Demmel. Computing accurate eigensystems of scaled diagonally dominant matrices. *SIAM Journal on Numerical Analysis*, 27:762–791, 1990.

[12] J.L. Barlow and S.L. Handy. The direct solution of weighted and equality constrained least squares problems. *SIAM Journal on Scientific and Statistical Computing*, 9:704–716, 1988.

[13] J.L. Barlow, P.A. Yoon, and H. Zha. An algorithm and a stability theory for downdating the ULV decomposition. Technical Report CSE-95-10, Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA, April 1995.

[14] J.L. Barlow, H. Zha, and P.A. Yoon. Stable chasing algorithms for modifying complete and partial singular value decompositions. Technical Report CSE-93-19, Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA, September 1993.

[15] W. Barth, R.S. Martin, and J.H. Wilkinson. Calculation of the eigenvalues of a symmetric tridiagonal matrix by the method of bisection. *Numerische Mathematik*, 9:386–393, 1967.

[16] J.J. Berbrands. On the relationships between SVD, KLT, and PCA. *Pattern Recognition*, 14:375–381, 1981.

[17] G. Bienvenue. Eigensystem properties of the sample space correlation matrix. In *Proceedings of ICASSP 83*, pages 332–335, 1983.

[18] Å. Björck. Stability analysis of the method of semi-normal equations for linear least squares problems. *Linear Algebra and Its Applications*, 88/89:31–48, 1987.

[19] Å. Björck. Least squares methods. In P. Ciarlet and J. Lions, editors, *Handbook of Numerical Analysis*, New York, 1989. Elsevier.

[20] Å. Björck, H. Park, and L. Eldén. Accurate downdating of least squares solutions. Preprint Series 947, Institute for Mathematics and its Applications, University of Minnesota, Minneapolis, MN, April 1992.

[21] A.W. Bojanczyk, R.P. Brent, P. Van Dooren, and F.R. De Hoog. A note on downdating the Cholesky factorization. *SIAM Journal on Scientific and Statistical Computing*, 8:210–221, 1987.

[22] A.W. Bojanczyk and J.M. Lebak. Downdating a ULLV decomposition of two matrices. In J.G. Lewis, editor, *Proceedings of the Fifth SIAM Conference on Applied Linear Algebra*, pages 261–265, Philadelphia, 1994. SIAM.

[23] N.K. Bose and H.M. Valenzuela. Recursive total least squares algorithm for image reconstruction from noisy, undersampled frames. *Multidimensional Systems and Signal Processing*, 4:253–268, 1993.

[24] J.R. Bunch. The weak and strong stability of algorithms in numerical linear algebra. *Linear Algebra and Its Applications*, 88/89:49–66, 1987.

[25] J.R. Bunch and C.P. Nielsen. Updating the singular value decomposition. *Numerische Mathematik*, 31:111–129, 1978.

[26] J.R. Bunch, C.P. Nielsen, and D. Sorensen. Rank-one modification of the symmetric eigenproblem. *Numerische Mathematik*, 31:31–48, 1978.

[27] P.A. Businger. Updating a singular value decomposition. *BIT*, 10:376–397, 1970.

[28] J.A. Cadzow. Spectral estimation and overdetermined rational model approach. In *Proceedings IEEE*, volume 70, pages 907–939. IEEE Computer Society Press, 1982.

[29] J. Carrier, L. Greengard, and V. Rokhlin. A fast adaptive multipole algorithm for particle simulations. *Numerische Mathematik*, 31:31–48, 1978.

[30] J. Carrier and V. Rokhlin. A fast algorithm for particle simulations. *Journal of Computational Physics*, 73:325–348, 1987.

[31] T.F. Chan. Rank-revealing QR factorizations. *Linear Algebra and Its Applications*, 88/89:67–82, 1987.

[32] J.M. Cioffi and T. Kailath. Windowed fast transversal filters, adaptive algorithms with normalization. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-33:607–625, 1985.

[33] S. Criveli and E.R. Jessup. Optimal eigenvalue computation on distributed-memory MIMD multiprocessor. Technical Report CU-CS-617-92, Department of Computer Science, University of Colorado, Boulder, CO, October 1992.

[34] C.E. Davila. Efficient recursive total least squares algorithm for FIR adaptive filtering. *IEEE Transactions on Signal Processing*, 42:268–280, 1994.

[35] J.W. Demmel and W.H. Kahan. Accurate singular values of bidiagonal matrices. *SIAM Journal on Scientific and Statistical Computing*, 11:873–912, 1990.

[36] J.W. Demmel and K. Veselic'. Jacobi's method is more accurate than QR. *SIAM Journal on Matrix Analysis and Applications*, 13:1204–1243, 1992.

[37] J.J. Dongarra, J.R. Bunch, C.B. Moler, and G.W. Stewart. *LINPACK User's Guide*. SIAM Publications, Philadelphia, 1979.

[38] J.J. Dongarra and D.C. Sorensen. A fully parallel algorithm for the symmetric eigenvalue problem. *SIAM Journal on Scientific and Statistical Computing*, 8:139–154, 1987.

[39] S.C. Eisenstat and I.C.F. Ipsen. Relative perturbation techniques for singular value problems. Technical Report YALEU/DCS/RR-942, Department of Computer Science, Yale University, July 1993.

[40] K.V. Fernando and B.N. Parlett. Accurate singular values and differential QD algorithms. Technical Report PAM-554, Center for Pure and Applied Mathematics, University of California, Berkeley, CA, July 1992.

[41] R.D. Fierro. Perturbation analysis for two-sided (or complete) orthogonal decompositions. Technical Report 94-02, Department of Mathematics, California State University, San Marcos, CA, February 1994.

[42] R.D. Fierro and J.R. Bunch. Bounding the subspaces from rank revealing two-sided orthogonal decompositions. Technical Report 94-05, Department of Mathematics, California State University, San Marcos, CA, May 1994.

[43] R.D. Fierro and P. C. Hansen. Approximating the LS and TLS solutions by rank revealing two-sided orthogonal decomposition. Technical Report 93-16, Department of Mathematics, University of California at Los Angeles, Los Angeles, CA, 1993.

[44] L.V. Foster. Rank and null space calculations using matrix decomposition without column interchanges. *Linear Algebra and Its Applications*, 74:47–71, 1986.

[45] F.Zhao and S.L. Johnsson. The parallel multipole method on the Connection Machine. *SIAM Journal on Scientific and Statistical Computing*, 12:1420–1437, 1991.

[46] P.E. Gill, G.H. Golub, W. Murray, and M.A. Saunders. Methods for modifying matrix factorizations. *Mathematics of Computation*, 28:505–535, 1974.

[47] G. H. Golub and W. Kahan. Calculating the singular values and pseudo-inverse of a matrix. *SIAM Journal on Numerical Analysis*, 2:205–224, 1965.

[48] G. H. Golub and C. Reinsch. Singular value decomposition and least squares solution. *Numerische Mathematik*, 14:403–420, 1970.

[49] G.H. Golub. Some modified matrix eigenvalue problems. *SIAM Review*, 15:318–344, 1973.

[50] G.H. Golub and C.F. Van Loan. *Matrix Computations, Second Edition*. The Johns Hopkins Press, Baltimore, MD, 1989.

[51] G.H. Golub and C.F. Van Loan. An analysis of the total least squares problem. *SIAM Journal on Numerical Analysis*, 17:883–893, 1980.

[52] W.B. Gragg and W.J. Harrod. The numerical stable reconstruction of Jacobi matrices from spectral data. *Numerische Mathematik*, 44:317–336, 1984.

[53] W.B. Gragg, J.R. Thorton, and D.D. Warner. Parallel divide and conquer algorithms for the symmetric tridiagonal eigenproblem and bidiagonal singular value problem. Technical report, Department of Mathematics, Naval Postgraduate School, Monterey, CA, 1992.

[54] M. Gu and S.C. Eisenstat. A stable and efficient algorithm for the rank-one modification of the symmetric eigenproblem. *SIAM Journal on Matrix Analysis and Applications*, 15:1266–1276, 1994.

[55] M. Gu and S.C. Eisenstat. A divide-and-conquer algorithm for the bidiagonal SVD. *SIAM Journal on Matrix Analysis and Applications*, 16:79–92, 1995.

[56] M. Gu and S.C. Eisenstat. A divide-and-conquer algorithm for the symmetric tridiagonal eigenproblem. *SIAM Journal on Matrix Analysis and Applications*, 16:172–191, 1995.

[57] M. Gu and S.C. Eisenstat. Downdating the singular value decomposition. Technical Report YALEU/DCS/RR-939, Department of Computer Science, Yale University, New Haven, CT, January 1993.

[58] W. W. Hager. *Applied Numerical Linear Algebra*. Prentice Hall, Englewood, NJ, 1988.

[59] S. Haykin. *Adaptive Filter Theory*. Prentice-Hall, Englewood Cliffs, NJ, 1986.

[60] T.L. Henderson. Geometric methods for determining system poles from transient response. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-29:982–989, 1981.

[61] N.J. Higham. A survey of condition number estimation for triangular matrices. *SIAM Review*, 29:575–598, 1987.

[62] Y.-P. Hong and C.-T. Pan. Rank-revealing QR factorizations and the SVD. *Mathematics of Computation*, 58:213–232, 1992.

[63] R.A. Horn and C.A. Johnson. *Matrix Analysis.* Cambridge University Press, New York, NY, 1985.

[64] S. Hosur, A.H. Tewfik, and D. Boley. Generalized ULV subspace tracking LMS algorithm. In M. Moonen and B. De Moor, editors, *SVD and signal processing III: algorithms, architectures, and applications*, pages 295–302, New York, 1995. Elsevier.

[65] Y. Huo and R. Schreiber. Efficient, massively parallel eigenvalue computation. Technical report, The Research Institute for Advanced Computer Science, NASA Ames Research Center, Moffet Field, CA, January 1993.

[66] K. Hwang and F.A. Briggs. *Computer Architecture and Parallel Processing.* McGraw-Hill, New York, 1984.

[67] E. R. Jessup and D.C. Sorensen. A parallel algorithm for computing the singular value decomposition. Technical Report MCS-TM-102, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, 1987.

[68] D.H. Johnson. The application of spectral estimation methods to bearing estimation problems. In *Proceedings IEEE*, volume 70, pages 1018–1028. IEEE Computer Society Press, 1982.

[69] T. Kato. *A Short Introduction to Perturbation Theory for Linear Operators.* Springer-Verlag, New York, 1982.

[70] C.L. Lawson and R.J. Hanson. *Solving Least Squares Problems*. Prentice-Hall, Englewood Cliff, NJ, 1974.

[71] T.J. Lee, F.T. Luk, and D.L. Boley. Computing the singular value decomposition on a fat-tree architecture. Technical Report 92-33, Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY, November 1992.

[72] C.E. Leiserson. Fat-trees: universal networks for hardware-efficient supercomputing. *IEEE Transactions on Computers*, C-24:892–901, 1985.

[73] K.J.R. Liu, D.P. O'Leary, G.W. Stewart, and Y.-J.J. Wu. An adaptive ESPIRIT based on URV decomposition. In *Proceedings of ICASSP 93*, pages IV–37–40, 1993.

[74] F.T. Luk and H. Park. On parallel Jacobi ordering. *SIAM Journal on Scientific and Statistical Computing*, 10:18–26, 1989.

[75] F.T. Luk and S. Qiao. A new matrix decomposition for signal processing. Technical Report TR 92-28, Department of Computer Science, Rensselaer Polytechnical Institute, Troy, NY, October 1992.

[76] M.P. Mahon. An alternative recursive updating scheme for the QR algorithm with application to adaptive beamforming. Master's thesis, Department of Electrical Engineering, The Pennsylvania State University, University Park, PA, 1991.

[77] M.P. Mahon, L.H. Sibul, and H.M. Valenzuela. Sliding window update for the basis matrix of the QR decomposition. *IEEE Transactions on Signal Processing*, 41:1951–1953, 1993.

[78] MathWorks Inc., Natick, MA. *MATLAB, High-performance Numeric Computation and Visualization Software.* 1993.

[79] M. Moonen, P. Van Dooren, and J. Vandewalle. An SVD updating algorithm for subspace tracking. *SIAM Journal on Matrix Analysis and Applications*, 13:1015–1038, 1992.

[80] M. Moonen, P. Van Dooren, and J. Vandewalle. Systolic array for SVD updating. *SIAM Journal on Matrix Analysis and Applications*, 14:353–371, 1993.

[81] M. Moonen, P. Van Dooren, and F. Vanpouche. On the QR algorithm and updating the SVD and URV decomposition in parallel. Preprint, IMA, 1992.

[82] D.P. O'Leary. The SVD and image restoration. In M. Moonen and B. De Moor, editors, *SVD and signal processing III: algorithms, architectures, and applications*, pages 315–322, New York, 1995. Elsevier.

[83] C.C. Paige. Error analysis of some techniques for updating orthogonal decompositions. *Mathematics of Computation*, 34:465–447, 1980.

[84] C.C. Paige and M.A. Saunders. Towards a generalized singular value decomposition. *SIAM Journal on Numerical Analysis*, 18:398–405, 1981.

[85] C.-T. Pan. A modification to the LINPACK downdating algorithm. *BIT*, 30:707–722, 1990.

[86] C.-T. Pan. A perturbation analysis on the problem of downdating a Cholesky factorization. *Linear Algebra and Its Applications*, 183:103–115, 1993.

[87] H. Park and L. Eldén. Downdating the rank-revealing URV decomposition. *SIAM Journal on Matrix Analysis and Applications*, 16:138–156, 1995.

[88] B. N. Parlett. The Rayleigh quotient iteration and some generalizations for non-normal matrices. *Mathematics of Computation*, 28:679–693, 1974.

[89] B.N. Parlett. *The Symmetric Eigenvalue Problem*. Prentice Hall, Englewood Cliffs, NJ, 1980.

[90] R. Ponnusamy, A. Choudhary, and A. Fox. Communication overhead on CM-5:an experimental performance evaluation. In H.J. Siegel, editor, *Proceedings of the Fourth Symposium on the Frontiers of Massively Parallel Computations*, pages 108–115. IEEE Computer Society Press, 1992.

[91] R. Roy and T. Kailath. ESPRIT – estimation of signal parameters via rotational invariance techniques. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37:984–995, 1989.

[92] R. Roy, P. Paulraj, and T. Kailath. ESPRIT – a subspace rotation approach to estimation of parameter of sinusoids in noise. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 34:1340–1342, 1986.

[93] H. Rutishauser. *On Jacobi rotation patterns*, pages 219–239. Amer. Math. Soc., 1963.

[94] R.O. Schmidt. *A signal subspace approach to multiple emitter location adn spectral estimation*. PhD thesis, Stanford University, Stanford, CA, 1981.

[95] R.O. Schmidt. Multiple emitter location and signal parameter estimation. *IEEE Transactions on Antennas and Propogation*, 34:276–280, 1986.

[96] L.H. Sibul. Application of singular value decomposition to adaptive beamforming. In *Proceedings of ICASSP 84*, 1984.

[97] D.C. Sorensen and P.T. Tang. On the orthogonality of eigenvectors computed by the divide-and-conquer techniques. *SIAM Journal on Numerical Analysis*, 28:1752–1775, 1991.

[98] G.W. Stewart. *Introduction to Matrix Computation.* Academic Press, New York, 1973.

[99] G.W. Stewart. The effects of rounding error on an algorithm for downdating a Cholesky factorization. *Journal of Institute of Mathematics and Its Applications*, 23:203–213, 1979.

[100] G.W. Stewart. Two simple residual bounds for the eigenvalues of a Hermitian matrix. *SIAM Journal on Matrix Analysis and Applications*, 12:205–209, 1991.

[101] G.W. Stewart. An updating algorithm for subspace tracking. *IEEE Transactions on Signal Processing*, 40:1535–1541, 1992.

[102] G.W. Stewart. Updating a rank-revealing ULV decomposition. *SIAM Journal on Matrix Analysis and Applications*, 14:494–499, 1993.

[103] G.W. Stewart. Updating URV decompositions in parallel. Technical Report CS-TR 2880, Department of Computer Science, University of Maryland, College Park, MD, April 1992.

[104] G.W. Stewart. On an algorithm for refining a rank-revealing URV decomposition and a perturbation theorem for singular values. Technical Report CS-TR 2626, Department of Computer Science, University of Maryland, College Park, MD, March 1991.

[105] M. Stewart and P. Van Dooren. An updating algorithm for on-line MIMO system identification. In M. Moonen and B. De Moor, editors, *SVD and signal processing III: algorithms, architectures, and applications*, pages 441–448, New York, 1995. Elsevier.

[106] G. Strang. *Linear Algebra and Its Applications.* Academic Press, New York, 1980.

[107] P.N. Swartztrauber. A parallel algorithm for computing the eigenvalues of a symmetric tridiagonal matrix. *Mathematics of Computation*, 60:651–668, 1993.

[108] J.A. Tague. *Estimation-correlation, modeling, and identification in adaptive array processors*. PhD thesis, The Pennsylvania State University, University Park, PA, 1987.

[109] J.A. Tague and L.H. Sibul. Estimator-correlator array processing: theoretical underpinnings and adaptive implementation. *Multidimensional Systems and Signal Processing*, 2:55–68, 1991.

[110] L. Tang. TTLS algorithms based on SVD and ULVD for solving ocean acoustic tomography. Master's paper, Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA, December 1995.

[111] Thinking Machines Corporation, Cambridge, MA. *Connection Machine CM-5 Technical Summary*. 1991.

[112] R.A. Thisted. *Elements of Statistical Computing*. Chapman and Hall, New York, 1988.

[113] D.W. Tufts and R. Kumaresan. Estimation of frequency of multiple sinusoids: making linear prediction perform like maximum likelihood. In *Proceedings IEEE*, volume 70, pages 975–1017. IEEE Computer Society Press, 1982.

[114] S. Van Huffel. *Analysis of the Total Least Squares Problem and its Use in Parameter Estimation*. PhD thesis, Katholieke Universiteit Leuven, Leuven, Belgium, 1987.

[115] S. Van Huffel and H. Park. Efficient reduction algorithms for bordered band matrices. Technical report, ESAT Laboratory, Katholieke Universiteit Leuven, Leuven, Belgium, 1992.

[116] S. Van Huffel and H. Park. Parallel reduction of bordered diagonal matrices. Technical Report 93-018, Army High Performance Computing Research Center, University of Minnesota, Minneapolis, MN, February 1993.

[117] S. Van Huffel and H. Park. Parallel tri- and bi-diagonalization of bordered bi-diagonal matrices. Technical Report 93-024, Army High Performance Computing Research Center, University of Minnesota, Minneapolis, MN, March 1993.

[118] S. Van Huffel and J. Vandewalle. *The Total Least Squares Problem: Computational Aspects and Analysis.* SIAM Publications, Philadelphia, 1991.

[119] S. Van Huffel and H. Zha. An efficient total least squares algorithm based on a rank revealing two-sided orthogonal decomposition. *Numerical Algorithms*, 4:101–133, 1993.

[120] C.F Van Loan. Generalizing the singular value decomposition. *SIAM Journal on Numerical Analysis*, 13:76–83, 1976.

[121] D.S. Watkins. *Fundamentals of Matrix Computations.* Wiley, New York, 1991.

[122] J.H. Wilkinson. *The Algebraic Eigenvalue Problem.* Oxford University Press, London, 1965.

[123] W. Wold, A. Ruhe, H. Wold, and W.J. Dunn, III. The collinearity in linear regression, the partial least squares PLS approach to generalized inverses. *SIAM Journal on Scientific and Statistical Computing*, 5:735–743, 1984.

[124] P.A. Yoon and J.L. Barlow. Modifying the singular value decomposition on the Connection Machine. In J.G. Lewis, editor, *Proceedings of the Seventh SIAM Conference on Parallel Processing for Scientific Computing*, pages 261–265, Philadelphia, 1995. SIAM.

[125] H. Zha. A two-way chasing scheme for reducing a symmetric arrowhead matrix to tridiagonal form. *Journal of Numerical Linear Algebra with Applications*, 1:49–57, 1992.

Appendix

Node Program for Bidiagonal Reduction
with Cyclic Storage Scheme

```
c
c       This node program will reduce the arrow-head matrix to
c       the bidiagonal matrix using cyclic storage scheme
c
c       Variables used:
c          abdz    -- contains values of a, b, d, and z in that order
c          myid    -- node id
c          idp1    -- myid + 1
c          idp2    -- myid + 2
c          idm1    -- myid - 1
c          idm2    -- myid - 2
c          hid     -- host id
c          cs      -- angles for the rotations
c          inrng1 -- processing the large block
c          inrng2 -- processing the small block
c

        .  special case -- first two rotations in each block
        .
100     continue
        again = .true.

50      continue
        ret = CMMD_receive_block(src, tag, op, isize)

150     continue
        go to (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19), op

1       ... op = 1

2       ... op = 2

3       continue
        ret = CMMD_send_block(idp1, tag, 7, isize)
```

```
      ret = CMMD_send_block(idp2, tag, 9, isize)
      ret = CMMD_receive_block(idp2, tag, cs, angsiz)
      abdz(3) = cs(2)*abdz(2)
      abdz(2) = cs(1)*abdz(2)
      op = 4
      it = it + 1
      go to 200

4     continue
      ret = CMMD_send_block(idp1, tag, 8, isize)
      ret = CMMD_swap(idp1, abdzt, bufsiz, abdz, bufsiz)
      call formrot(abdzt(2), abdz(3), cs(1), cs(2))
      abdz(1) = cs(1)*abdz(1)
      call applyrot(abdzt(1), abdz(2), cs(1), cs(2), 1)
      op = 3
      it = it + 1
      go to 200

5     ... op = 5

6     ... op = 6

7     continue
      ret = CMMD_receive_block(idp1, tag, cs, angsiz)
      call applyrot(abdz(2), abdz(1), cs(1), cs(2), 1)
      it = it + 1
      go to 100

8     continue
      if (inrng1 .and. it .eq. maxit) then
         ret = CMMD_send_block(hid, tag, 20, isize)
      end if
      ret = CMMD_swap(idm1, abdzt, bufsiz, abdz, bufsiz)
      call formrot(abdz(2), abdzt(3), cs(1), cs(2))
      abdz(3) = cs(2)*abdzt(1)
      call applyrot(abdz(1), abdzt(2), cs(1), cs(2), 1)
      it = it + 1
      go to 100

9     continue
      call formrot(abdz(1), abdz(3), cs(1), cs(2))
      ret = CMMD_send_block(idm2, tag, cs, angsiz)
      ret = CMMD_send_block(idm1, tag, cs, angsiz)
      it = it + 1
      go to 100
```

```
10     ...  terminate
       go to 999

200    continue
       if (again) then
          again = .not. again
        go to 150
       else
          if (myid .ne. 0) then
             ret = CMMD_send_block(idm1, tag, op, isize)
          else
             ret = CMMD_send_block(hid, tag, op, isize)
          end if
          go to 100
       end if

999    continue
       .
       .  similarly for the small block
       .
```